George Danezis—Ed. (UCL)
Vasilios Mavroudis (UCL)
Sebastian Meiser (UCL)
Ania Piotrowska (UCL)
Helger Lipmaa (UT)
Michał Zając (UT)
Claudia Diaz (KUL)
Tariq Elahi (KUL)
Benjamin Weggenmann (SAP)
Aggelos Kiayias (UEDIN)

Design, modelling and analysis

Deliverable D3.2

December 15, 2017
PANORAMIX Project, # 653497, Horizon 2020
http://www.panoramix-project.eu

Horizon 2020
European Union funding
for Research & Innovation

Revision History

Revision	Date	Author(s)	Description
0.1	2017-04-09	Sebastian Meiser	Initial Deliverable compilation.
0.2	2017-04-10	George Danezis	Internal WP3 UCL Review.
0.3	2017-04-25	Benjamin Weggenmann	External Review (SAP).
0.4	2017-04-27	Mirjam Wester	Final Review by Coordinator.
0.5	2017-04-27	Sebastian Meiser	Final WP3 UCL Review.
1.0	2017-04-28	Mirjam Wester	Final version submitted to EC.
1.1	2017-11-10	Vasilios Mavroudis	Revision after second periodic review.
2.0	2017-12-15	Mirjam Wester	Final revised version submitted to EC.

Executive summary

Deliverable D3.2 summarizes the research & development contributions of the PANORAMIX project partners involved in Work Package 3 (WP3) over months M10–M20 of the project. The contributions relate to better techniques to prove the correctness of private shuffles that are key to implementing electronic voting systems; novel techniques for implementing performant low-latency decryption mix networks using cover traffic and active defenses to provide robustness; techniques to mitigate long-term key compromises, even against very powerful adversaries that could compromise supply chains and the hardware platform of mix nodes; and finally modern techniques to define and quantify anonymity inspired from differential privacy and modern game-based cryptographic definitions.

In terms of performant Non-Interactive Zero Knowledge proofs and arguments for shuffling secret messages (or ballots) we present novel techniques that outperform the existing state of the art and allow for more scalable private electronic elections. Proofs of correct shuffling for 100,000 ballots can be produced in fewer than 2 minutes on a commodity PC, and can be checked by any interested party in less than 3 minutes.

In relation to decryption mix networks we present the design of the "Loopix" system, that supports low-latency anonymous communications by using well controlled cover traffic to confuse traffic analysis. We also use cover traffic to detect active dropping attacks aiming to reduce anonymity, allowing the system to act to minimize their impact. Furthermore, we present the "Myst" architecture to protect long-term mix node decryption keys from compromise, even against very powerful adversaries that could compromise the hardware supply chains of mix nodes. Myst uses an array of independently sourced, inexpensive, tamper-resistant components to protect keys even when the mix node, and some of the components are malicious. We also present the design and research of our decryption mix network libraries "sphinxmix" that are available as free software.

Finally, we leverage a novel game-based set of definitions for anonymity systems to quantify the security of the established Tor system even if parts of the Internet's infrastructure are corrupt. The underlying definition follows modern conventions in cryptography where privacy is abstracted as an indistinguishability game between two possible worlds that an adversary cannot easily distinguish between. We prove that our calculations are sound and, under reasonable assumptions, tight.

Contents

1. Introduction

The objective of PANORAMIX is to leverage the versatile concept of mix networks to develop a multipurpose infrastructure for anonymous communication and to integrate this infrastructure into practical and impactful applications. Mix networks hide the identity of senders and receivers of messages by sending these messages in an encrypted form through a series of mix servers that shuffle the messages to hinder the linkability of senders and receivers. PANORAMIX focuses on three diverse applications: (1) private electronic voting protocols (WP5); (2) privacy-friendly statistics and big data gathering protocols (WP6); and (3) privacy-preserving messaging protocols (WP7).

Work Package 3 (WP3) and this Deliverable D3.2 focus on modelling and analyzing mix networks, comprising the development of novel, secure and efficient mix network designs and the theoretical and experimental security analysis of mix networks. It supports the more applied development efforts, which are part of Work Package 4 (WP4), and provides well evaluated design options, that the core mix implementation, as well as applications may use. WP3 in that sense is the "R" (Research) in the "R&D" effort underpinning the PANORAMIX project.

This document presents an intermediate report of the research progress after the first two stages of PANORAMIX at Month M20 of the project, closely following the project proposal. In the following section we outline the structure of this deliverable and describe the direct relation of the outlined sections to the tasks described in the project proposal. Each further chapter then describes in detail our progress on the respective task.

1.1 Outline of the deliverable

The deliverable D3.2 as described in the project proposal comprises:

> Deliverable D3.2 (Interim report) [M20] Modelling and design elements.
> *Describes:*
>
> - *a first iteration of a NIZK shuffle proof that may be used in implementation within WP5* (WP3.2),
> - *integrated robustness into efficient mix network designs and decryption mixes* (WP3.1),
> - *robust definitions of mix networks as differentially private mechanisms* (WP3.3).

The structure of this Deliverable D3.2 intentionally follows the structure defined in the project proposal. Part I deals with "a first iteration of a NIZK shuffle proof that may be used in implementation within WP5"; Part II presents, in two chapters, design elements that "integrate robustness into efficient mix network designs and decryption mixes"; and finally Part III presents "robust

definitions of mix networks as differentially private mechanisms".

1.2 WP3 tasks and mapping to Deliverable D3.2

In this section, we relate each chapter of this deliverable to each of the tasks of WP3, and summarize their key contributions to the PANORAMIX project. We emphasize (**in bold**) using the definitions of WP3 Tasks, as they are defined in the project proposal, which parts of each task the current deliverable covers; and summarize the key contributions of each chapter of the deliverable at the end.

1.2.1 Part I – NIZK shuffle proofs (Task 3.2)

Background. As a general pattern in security and cryptographic engineering, to obtain provable security against malicious participants (i.e., parties who are not honest, and can arbitrarily deviate from the expect behavior), one uses zero knowledge (ZK) proofs. Those allow anyone to verify the correct function of other participants without requiring them to learn any secrets – concurrently supporting high-confidentiality (privacy) and high-integrity (correctness) security properties.

More precisely, a ZK proof is a protocol between two participants, a prover and a verifier (that can be instantiated by parties, relevant in the concrete situation), that satisfies the following three requirements:

- Completeness: an honest verifier accepts an honest prover (e.g., if the shuffle was performed correctly, the proof is accepted),
- Soundness: the probability that an honest verifier accepts a dishonest prover is negligible (e.g., if the shuffle was not done correctly, the prover will be caught almost always), and
- Zero knowledge: a proof by an honest prover leaks no information except that the proved statement holds true (e.g., the verifier only becomes convinced that the shuffle was performed correctly, without obtaining any additional information about, say, the plaintexts or the used permutation).

In the context of PANORAMIX, and mix networks in general, a ZK proof of a shuffle enables a server to prove to the verifier that she has performed a shuffle correctly without revealing any side information. In the case of the re-encryption shuffle, this means that two lists of plaintexts commit to the same multi-set of plaintexts. Similarly, one can construct ZK proofs of decryption shuffles – however, those proofs can be inefficient unless carefully chosen decryption mix network designs are chosen.

ZK proofs can be interactive and in this case every prover has to interact with every possible verifier and the proofs are not transferable. ZK proofs can also be made non-interactive and are called NIZK. In this case, a prover just creates a single shuffle proof that can be later transferred to all interested verifiers who can verify its correctness even after the prover has gone offline. This allows the correct operation of the mix network to be "universally verifiable" – providing high, and first hand assurance that no messages were inserted, modified or dropped by mixes.

A shuffle together with a NIZK shuffle proof is the most secure version of a shuffle needed in practice. However, this high level of security comes with an efficiency cost. For example, pre-existing ZK shuffle proofs are somewhat less efficient (e.g., requiring approximately ten exponentiations per a shuffled item) than randomized partial checking. Moreover, NIZK shuffle proofs work either in the "random oracle" (RO) model or in the common reference string (CRS) model. A large body of

research deals with the construction of efficient NIZK shuffle proofs. There are literally dozens of papers that construct more and more efficient NIZK shuffle proofs in the RO model. Proofs exist that claim to be able to shuffle 100,000 ciphertexts in 2 minutes. However, one can argue that such proofs are either insufficiently efficient or just very complicated (to understand, to implement, to explain).

Definition of Task 3.2.2. During this subtask, we will aim to **propose more efficient (or as efficient, but conceptually simpler) shuffle proofs** as known in the literature. Depending on the outcomes of task 3.1.1, the protocols will be either in the random oracle model or in the **common reference string model**. We will also compare the efficacy of resulting NIZK shuffle proofs with the best existing randomized partial checking methods, to see whether NIZK shuffle proofs (or randomized partial setting) should be used in WP5/6/7. Thus, **the output of this task will be used in other work packages, especially WP5, and possibly implemented**. This task will provide necessary cryptographic background also for other packages. Another interesting aspect of many existing NIZK shuffle proofs is that they seem to be highly parallelizable. However, this direction has not been investigated much, and in particular we are not aware of any shuffle proofs that are designed to be efficient on modern parallel architectures (e.g., graphics processors). Construction of efficiently parallelizable NIZK shuffle proofs will be one of the goals of the current task. This subtask starts in parallel with subtask 3.2.1. The main deliverable of this subtask will be **a description of an initial variant of a concrete NIZK shuffle proof** that can be used then in at least WP5.

Summary of contributions of Deliverable D3.2. Our key advances relating to this task are presented in **Chapter 2** – "Efficient non-interactive zero-knowledge shuffles", where we present a novel NIZK for a shuffle of a re-encryption mix network that outperforms previous proposals. As outlined in the Task description, the novel techniques provide more efficient shuffle proofs, including in the CRS model; and they are evaluated and tuned in the context of electronic elections (WP5); and they have been considered as candidates for the work in WP4.

As summarized in the chapter, the new technique allows the processing (i.e. mixing, re-encryption and production of a proof) of almost 100,000 ciphertexts in about 2 minutes. The Verification time of the proof for the same number of messages is under 3 minutes. These benchmarks were computed on a standard PC (i5 processor and 8 GB of RAM), and performance would be much better when the computations were done on a server, which indeed is the case for e-voting. The full cryptographic details of the novel technique, comparison with previous work, and security arguments are provided in **Chapter 2**.

1.2.2 Part II – Robust and efficient mix network designs (Task 3.1)

Background. Mix networks are specialized routers that cryptographically hide the correspondences between input and output messages, and obscure their timings through batching, delaying, dropping or adding dummy traffic, in order to obscure who is talking to whom. A number of architectures for mix nets have been discussed in the literature, and two of the major ones are decryption mix nets – that accept ciphertexts, decrypt them and forward them – and the re-encryption mix nets – that re-blind their inputs before revealing them.

Task 3.1 researches and evaluates design options for decryption mix networks. Those rely on a sender encrypting a message multiple times to a sequence of keys, and then forwarding the encoded message to a sequence of mixes. Each mix decrypts a layer of the message, checks it for integrity violations, and forwards the message to its next destination. Decryption mix networks

have a number of advantages over re-encryption mix networks: they allow flexible selection of the intermediary mixes, and also more complex mixing strategies. This is due to the fact that the sender may include arbitrary routing information destined to each intermediate mix alongside the message to be anonymized. Secondly, decryption mix networks can relay an arbitrarily long (yet fixed size) message, and only require a single public key operation per mixing operation per message. This feature makes them a good choice for PANORAMIX use cases WP7, relating to messaging applications, and more flexible for WP6, relating to collection of private statistics.

Decryption mix networks have, however, a number of drawbacks. First, zero-knowledge proofs of shuffling are very inefficient. This exposes those mix networks to active attacks by the infrastructure, and dishonest mix nodes, that could be used to facilitate traffic analysis and tracing. Secondly, designs for decryption mix networks that offer facilities for replies, and efficient key management, require the use of longer term decryption keys. These keys are extremely high-value, since their exposure to the adversary could facilitate de-anonymization even in the future, using intercepted material. Thus, robust protections need to be used by the mixing protocols or the implementation of mix nodes to safeguard those keys.

Task 3.1.2 is concerned with studying in depth decryption mix nets with a view to extending them for high-reliability delivery of election ballots (to support WP5). We hypothesize that randomized partial checking (RPC) techniques, or other "cut-and-chose" mechanisms, could be adapted. However, we note that there are complexities associated with implementing RPC in asynchronous settings, and a secure infrastructure needs to be devised to support challenges in the context of decryption. Decryption mix nets also provide ways to **route replies to anonymous messages**. Prior work has looked at making such **replies indistinguishable from other messages, however suffers from a number of shortcomings we plan to address**. Replies **require mixes to retain long term confidentiality keys, which preclude any forward secrecy guarantees**. Alternative designs could instead, for example, maintain state in routers to facilitate replies; alternatively pick-up points (nym-servers) can be developed to operate with only ephemeral reply channels. While some draft designs have previously appeared, this project is the first to seriously consider such extensions to decryption mix nets, and will develop techniques specifically tailored to surveying, big data and statistics applications to support WP6. Mix network routing strategies and network topology impacts both the level of privacy offered as well as its resilience in the face of network failures or attacks. We will **study the design space of network topologies and routing algorithms, and identify the most suitable designs for the selected use cases**. Finally, we will **develop cover traffic strategies to provide advanced properties such as unobservability (meaning that it is not possible for an adversary to determine whether or not a user is sending or receiving a message). Cover traffic comes at the cost of processing and network overhead, and thus, our focus will be on developing strategies that maximize protection for a given tolerable overhead.**

Summary of contributions of Deliverable D3.2. The key contributions to Task 3.1 are presented in **Chapter 3** – "The Loopix Anonymity system" and **Chapter 4** on "Strong and robust security guarantees in the presence of malicious components". A key WP3 contribution to the development effort is the "graduation" of our codebase implementing the core of a decryption mix network from pure research to development and is summarized in the **Appendix A** – "The Sphinxmix Python Package".

The Loopix Anonymity system is a research prototype embodying, exploring and evaluating key ideas of PANORAMIX decryption mix networks: it supports replies to anonymous messages, and a flexible mix network routing structure that supports email or instant messaging providers

of direct interest to partners working on messaging as part of WP7. Instead of relying primarily on delaying messages to provide mixing and traffic analysis resistance, it uses a controlled amount of cover traffic to confuse passive and active adversaries as to the real destination of messages – a careful evaluation of the cost versus benefit of using covert traffic is provided in the chapter. Finally, instead of relying on NIZK or RPC for robustness – something that is not appropriate for messaging applications due to natural network message dropping to deal with congestion – it employs an alternative novel technique for detecting malicious active attacks. Loops of cover traffic are injected by mix nodes and clients to detect whether they are under active attacks, and to mitigate them. Full performance details are provided in **Chapter 3**, and the research prototype of the system is available publicly on a code repository as open source software[1].

A further contribution to Task 3.1 relates to the risk of exposing long term decryption keys in case mixes are compromised at some point. As part of **Chapter 4**, we describe techniques for a hardware key management that protects the confidentiality of decryption keys (and signing keys) even against adversaries that may have compromised some hardware components the mixes rely upon. The case of supply-chain attacks and hardware compromise is the harshest we can consider, since mix nodes cannot even rely on their own hardware platforms to ensure the confidentiality of keys – and so far protecting again hardware trojans and other supply-chain attacks was considered an open research problem with applications beyond mix networks. In **Chapter 4**, we present the "Concordia" architecture that makes use of multiple independently sourced inexpensive tamper-resistant hardware components, that collectively allow a mix to generate random numbers, perform decryptions and sign messages – without ever exposing long term keys to any single component or the mix network itself. We present the architecture and note that it is very well suited to decryption mix networks: our research hardware prototype allows for processing 315 messages a second, per quorum of components and can be parallelized to scale arbitrarily. We propose this architecture to mediate the risks posed by the lack of perfect forward secrecy in decryption mix network designs.

Finally, in **Appendix A** we present a brief overview of our research libraries for decryption mix nets, that we have successfully open sourced and provided as the core for the decryption mix network in WP4 – to be used by all partners. The SphinxMix Python library is open source[2], available through a standard Python package, well documented, tested as part of a continuous integration regime, and has near 100% test coverage. A summary of its architecture and its current documentation is provided in **Appendix A**.

1.2.3 Part III – Robust security definitions for mix networks (Task 3.3)

Background. A number of definitions and metrics for privacy have been proposed over the years, but all have serious shortcomings. Definitions based on counting the number of possible actors ignore their relative likelihoods, while established entropy based metrics conflate the information leaked by the mechanism with the prior or side information known to an adversary. Neither of those composes well, making it difficult to specify the security of a mix network when used multiple times to route correlated traffic. The objective of Task 3.3 is the study of privacy definitions and metrics that precisely measure the privacy loss through the use of a mix net, in a way that is independent to adversary side information, and composes well to estimate the privacy provided by multiple uses of a mix network.

[1] See `https://github.com/UCL-InfoSec/loopix`
[2] See `https://github.com/UCL-InfoSec/sphinx`

Task 3.3.2. We will choose from existing designs of differential privacy algorithms that can meet our new definitions. Specifically, differential privacy mechanisms proceed by **introducing suitable noise in the data that partially randomizes this data so that a minimal amount of information is leaked to anybody that possesses the database while the utility of the data is preserved** (at least within reason). Differential privacy algorithms exhibit natural usability / privacy trade-offs that we will explore and exploit in our designs. We note that besides **differential privacy being used as a definition of privacy for mix nets, anonymous channels in general can also become a component of larger privacy preserving systems**. In particular, such private channels may be used to improve the performance of privacy mechanisms for surveys and statistics collection, by lowering the amount of noise that needs to be added to the results through partially hiding its true originator. Task 3.3.3 will explore the full potential of this insight, by adapting known important privacy mechanisms, e.g. statistical or relational databases, Private Information Retrieval, Oblivious RAM, and differentially private data collection, to using mix nets as part of their mechanism. We conjecture that the added adversary confusion about the origin of each action will allow us to significantly lower the cost of privacy protections.

Summary of contributions of Deliverable D3.2. The key contributions to Task 3.3 are presented in **Chapter 5** – "Computing tight anonymity bounds for Tor against malicious network infrastructure". Our analysis leverages the AnoA definitions for anonymous communication that are based on differential privacy to analyze the degree of anonymity the Tor network provides in lieu of a corrupt network infrastructure. Our analysis establishes a robust base-line for the degree of anonymity a mix network should provide if observers of parts of the Internet, such as autonomous systems, Internet exchange points and landing points of submarine cables are trying to break the anonymity provided by the system. The definitions we leverage for our analysis are inspired by differential privacy and allow for a comparison between novel mix networks and Tor. Moreover, our evaluation shows that the current state of the art protocol Tor is susceptible to an observing top-tier provider, facing a reduction of anonymity of up to 27.8% for a single company (Level 3). A collusion of three companies (Level 3, NTT and DTAG) even reduces anonymity by 47.2%.

Part I

NIZK shuffle proofs

2. Efficient non-interactive zero-knowledge shuffles

2.1 Introduction

A typical application of mix networks is in e-voting, where each voter (assume that there are n of them) encrypts his ballot by using an additively homomorphic public-key cryptosystem, and sends it to the bulletin board. After the vote casting period has ended, the bulletin board (considered to be the 0th, non-mixing, mix server) forwards all encrypted ballots to the first mix server. A small number (say, M) of mix servers are ordered sequentially. The kth mix server obtains a tuple $\vec{\mathfrak{v}}$ of input ciphertexts from the $(k-1)$th mix server, shuffles them, and sends a tuple $\vec{\mathfrak{v}'}$ of output ciphertexts to the $(k+1)$th mix server. Shuffling means that the kth mix server generates a random permutation $\sigma \leftarrow_r S_n$ and a vector $\vec{s}$ of randomizers, and sets $\vec{\mathfrak{v}}'_i = \vec{\mathfrak{v}}_{\sigma(i)} \cdot \mathsf{enc}_{\mathsf{pk}}(0; s_i)$, where $\mathsf{enc}_{\mathsf{pk}}(m; r)$, given a public key pk, a message m, and a randomizer r (from some randomizer space $\mathcal{R}$), returns a ciphertext, i.e. vector $\vec{\mathfrak{v}}'_i$ is a vector of permuted and rerandomized ciphertexts $\vec{\mathfrak{v}}$. The last mix server (the $(M+1)$th one, usually implemented by using multi-party computation) is again a non-mixing server, who instead decrypts the results.

A mix network typically preserves the anonymity of voters, if at least one of the participating mix servers is honest. To achieve security against an active attack (where some of the shuffles were not done correctly) is more difficult. In a nutshell, each server should prove in zero knowledge [22] that her shuffle was done correctly, i.e., prove that there exists a permutation σ and a vector $\vec{s}$, such that $\vec{\mathfrak{v}}'_i = \vec{\mathfrak{v}}_{\sigma(i)} \cdot \mathsf{enc}_{\mathsf{pk}}(0; s_i)$ for each i. The resulting zero-knowledge proof is usually called a *(zero-knowledge) shuffle argument*.

Moreover, to obtain active security of the whole mix network, it is important that the outputs of incorrect shuffles are ignored. This means that each mix server (including the $(M+1)$th one) has to verify the correctness of each previous mix server, and only apply her own shuffle to the output of the (multi-)shuffle where each previous server has been correct. Intuitively, this means that the verification time is the real bottleneck of mix networks.

Substantial amount of work has been done on interactive zero-knowledge shuffle arguments. Random oracle model shuffle arguments are already quite efficient, see, e.g., [24]. However, an ever-growing amount of research [10, 21, 33, 6] has provided evidence that the random oracle model yields properties that are impossible to achieve in the standard model. (See [12] for recent progress on NIZK arguments in the random oracle model.)

Much less is known about shuffle arguments in the common reference string (CRS, [7]) model, without using random oracles. This chapter describes two recent results provided by the PANORAMIX project working group. The first result by Fauzi and Lipmaa [17] brought great efficiency gain for both prover and verifier. The later scheme, by Fauzi, Lipmaa and Zając [18] made shuffles even more efficient for the verifier with a small loss in the prover efficiency.

2.1.1 Shuffles as a part of the PANORAMIX project

The design of efficient shuffle arguments is an important part of WP3. Through WP3 we will disseminate to more use-case oriented packages like WP 5 (e-voting). The requirement for new shuffle arguments can be easily motivated twofold. The argument itself is crucial for providing privacy of the mix network, especially in the use case of e-voting. On the other hand, known solutions are usually not efficient enough. For a network where the number of messages to mix goes into hundreds of thousands or even millions, new arguments were a must. Otherwise the whole system would become inefficient with shuffling appointed as a bottleneck, what would compromise usability of the system in a great manner.

As a technique to provide robust and private mixing, shuffle arguments are crucial for Tasks 3.1.1 (notions of unlinkability and anonymity) and 3.1.3 (security evaluation). Development of this primitive was stated as of independent interest in Task 3.2. We justify the model we used as it is stated in Task 3.2.1; propose shuffle arguments as required in Task 3.2.2. Furthermore, we validate our results (Task 3.2.3) – both proposed arguments were accepted to top-notch cryptographic conferences: the argument by Fauzi and Lipmaa to CT-RSA and the argument by Fauzi, Lipmaa and Zając to Asiacrypt. Task 3.3, focused on making mix networks more efficient will benefit from the below results as well. We emphasize that proposed protocols are currently the fastest CRS-based shuffle arguments (we motivated using this model above).

Knowledge gained during designing these arguments has been proven useful in WP 5 (Use case: e-voting) Tasks: 5.1 (Requirement analysis and specification), 5.2 (Design), 5.3 (Validation and product implementation). The arguments make it possible to process (i.e. mix, re-encrypt and provide a proof) almost 100,000 ciphertexts in about 2 minutes. Verification time for such an amount of messages is under 3 minutes. Both of these numbers were computed on a standard PC with i5 processor and 8 GB of RAM. We can easily assume that these numbers would be much better if the computations are done on a server, what indeed is the case for e-voting. Furthermore, since the numbers are not large, verification of the whole e-voting process could be processed on a PC.

2.1.2 Fauzi-Lipmaa shuffle argument

In *Efficient Culpably Sound NIZK Shuffle Argument without Random Oracles* [17] Fauzi and Lipmaa proposed a new pairing-based NIZK shuffle argument in the CRS model. Differently from [29], they proved the culpable soundness of the new argument instead of white-box soundness. Compared to [26], which also achieves culpable soundness, the new argument has 3 times faster proving and more than 4 times faster verification. Compared to [26, 29], it is based on a more standard cryptosystem (ElGamal). While the new shuffle argument is still at least 2 times slower than the most efficient known random oracle based shuffle arguments, it has almost optimal *online* prover's computation. Of course, a full efficiency comparison can only be made after implementing the different shuffle arguments.

The construction works as follows. First they commit to the permutation σ (by committing separately to first $n-1$ rows of the corresponding permutation matrix $\vec{A}$) and to the vector $\vec{t}$ of blinding randomizers. Here, they use the *polynomial commitment scheme* (see Sect. 2.2) with $\mathsf{com}(\mathsf{ck};\vec{m};r) = (\mathfrak{g}_1,\mathfrak{g}_2^{\gamma})^{rP_0(\chi)+\sum_{i=1}^{n} m_i P_i(\chi)} \in \mathbb{G}_1 \times \mathbb{G}_2$, in pairing-based setting, where $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear pairing, g_i is a generator of $\mathbb{G}_i$ for $i \in \{1,2\}$, $(P_i(X))_{i=0}^{n}$ is a tuple of linearly independent polynomials, χ is a trapdoor, γ is a knowledge secret, and $\mathsf{ck} = ((\mathfrak{g}_1,\mathfrak{g}_2^{\gamma})^{P_i(\chi)})_{i=0}^{n}$ is the CRS. For different values of $P_i(X)$, variants of this commitment scheme have been proposed before [23, 25, 28].

They show that $\vec{A}$ is a correct permutation matrix by constructing n witness-indistinguishable succinct *unit vector arguments*, each of which guarantees that a row of $\vec{A}$ is a unit vector, for implicitly constructed $\vec{A}_n = \vec{1_n} - \sum_{i=1}^{n-1} \vec{A}_i$. They use the recent square span programs (SSP, [13]) approach to choose the polynomials $P_i(X) = y_i(X)$ so that the unit vector argument is efficient. Since unit vectors are used in many contexts, this argument is of independent interest.

After that, they postulate a natural concrete verification equation for shuffles, and construct the shuffle argument from this. If privacy were not an issue (and thus $\mathfrak{v}'_i = \mathfrak{v}_{\sigma(i)}$ for every i), the verification equation would just be the tautology $\prod_{i=1}^{n} \hat{e}(\mathfrak{v}'_i, \mathfrak{g}_2^{y_i(\chi)}) =^? \prod_{i=1}^{n} \hat{e}(\mathfrak{v}_i, \mathfrak{g}_2^{y_{\sigma^{-1}(i)}(\chi)})$. Clearly, if the prover is honest, this equation holds. However, it does not yet guarantee soundness, since an adversary can use $\mathfrak{g}_1^{y_j(\chi)}$ (given in the CRS) to create $(\mathfrak{v}'_i)_{i=1}^n$ in a malicious way. To eliminate this possibility, by roughly following an idea from [26], they also verify that $\prod_{i=1}^{n} \hat{e}(\mathfrak{v}'_i, \mathfrak{g}_2^{\hat{y}_i(\chi)}) =^? \prod_{i=1}^{n} \hat{e}(\mathfrak{v}_i, \mathfrak{g}_2^{\hat{y}_{\sigma^{-1}(i)}(\chi)})$ for some well-chosen polynomials $\hat{y}_i(X)$. (Instead of n univariate polynomials, [26] used n random variables χ_i, increasing the size of the secret key to $\Omega(n)$ bits.)

To show that the verifications are instantiated correctly, they also needed a *same-message argument* that shows that commitments w.r.t. two tuples of polynomials $(y_i(X))_{i=1}^n$ and $(\hat{y}_i(X))_{i=1}^n$ commit to the same plaintext vectors. They construct an efficient same-message argument by using an approach that is (again, roughly) motivated by the QAP-based approach of [20]. This argument is an argument of knowledge, given that the polynomials $\hat{y}_i(X)$ satisfy an additional restriction.

Since also privacy is required, the actual verification equations are more complicated. In particular, $\mathfrak{v}'_i = \mathfrak{v}_{\sigma(i)} \cdot \mathsf{enc}_{\mathsf{pk}}(1; t_i)$, and (say) $\mathfrak{g}_2^{y_{\sigma^{-1}(i)}(\chi)}$ is replaced by the second element $\mathfrak{g}_2^{\gamma(r_i y_0(\chi) + y_{\sigma^{-1}(i)}(\chi))}$ of a commitment to $\vec{A}_i$. The resulting complication is minor (it requires one to include into the shuffle argument a single ciphertext $U \in \mathbb{G}_1^2$ that compensates for the added randomness). The full shuffle argument consists of commitments to $\vec{A}$ and to $\vec{t}$ (both committed twice, w.r.t. the polynomials $(y_i(X))_{i=0}^n$ and $(\hat{y}_i(X))_{i=0}^n$), n unit vector arguments (one for each row of $\vec{A}$), $n-1$ same-message arguments, and finally U.

If $\hat{y}_i(X)$ are well-chosen, then from the two verification equations and the soundness of the unit vector and same-message arguments it follows, under a new computational assumption PSP (*Power Simultaneous Product*, related to an assumption from [26]), that $\mathfrak{v}'_i = \mathfrak{v}_{\sigma(i)}$ for every i.

They prove culpable soundness [26, 27] of the new argument. Since the security of the new shuffle argument does not depend on the cryptosystem either having knowledge components or being lifted, they used ElGamal encryption [15] instead of the non-standard knowledge BBS encryption introduced in [29]. Since the cryptosystem does not have to be lifted, one can use more complex voting mechanisms with more complex ballots. The use of knowledge assumptions means that the new argument is an argument of knowledge.

The new shuffle argument can be largely precomputed by the prover and forwarded to the verifier even before the common input (i.e., ciphertexts) arrive. Similarly, the verifier can perform a large part of verification before receiving the ciphertexts. (See [38] for motivation for precomputation.) The prover's computation in the online phase is dominated by just two $(n+1)$-wide multi-exponentiations (the computation of U). The multi-exponentiations can be parallelized; this is important in practice due to the wide availability of highly parallel graphics processors.

2.1.3 Fauzi-Lipmaa-Zając shuffle argument

Here we describe the recent result by Fauzi, Lipmaa, Zając *A Shuffle Argument Secure in the Generic Model* ([18]). In the results mentioned above [26, 30, 17], the authors based the soundness of their shuffle argument on some novel hardness assumptions, and then proved that the assumptions are

secure in the generic bilinear group model (GBGM). It seems to be an obvious question whether one can obtain some efficiency benefit by bypassing the intermediate assumption and proving the soundness of the shuffle argument directly in the GBGM. Fauzi et al. (2016) show this is indeed the case. They improve on the efficiency of the previous CRS-based shuffle arguments by proving the security of the protocol in the GBGM and *without* using knowledge assumptions. Due to the use of GBGM, they first define a sensible security model.

First, recall that in the GBGM, the adversary inputs some group elements $\mathfrak{G}_i = \mathfrak{g}^{\chi_i}$, where $\mathfrak{g}$ is a group generator and χ_i are various (not necessarily independent) random values. One assumes that each group element $\mathfrak{H}_j$ output by the adversary is of the form $\mathfrak{H}_j = \mathfrak{g}_z^{F_j(\vec{\chi})}$, where $F_j(\vec{X})$ are known linear polynomials and $\mathfrak{g}_z$ is a generator of the group $\mathbb{G}_z$, $z \in \{1, 2\}$.

One philosophical question when using the GBGM is what exactly is the input of the adversary. In the intended usage cases, the shuffle argument is a part of a mix network. Clearly, the mix network should remain secure against coalitions between parties (in the case of e-voting, either voters, or some of the mix servers themselves) that create the input ciphertexts, and parties who perform the shuffling. It is a common practice to model such coalitions as a single adversary. In the GBGM, it is natural to model this single adversary — who may corrupt everybody who has produced any part of the input to the verifier — as a generic adversary. This means that an adversary, who has generated a (say, SXDH [16]) ciphertext $\vec{\mathfrak{v}}_i = (\mathfrak{v}_{i1}, \mathfrak{v}_{i2}, \mathfrak{v}_{i3})$, knows polynomials $V_{ij}(\vec{X})$ and $V'_{ij}(\vec{X})$, such that $\log \mathfrak{v}_{ij} = V_{ij}(\vec{\chi})$ and $\log \mathfrak{v}'_{ij} = V'_{ij}(\vec{\chi})$. This is somewhat similar to the approach taken in [30] who used knowledge assumptions to then obtain the random variables — more precisely, plaintexts and randomizers — hidden in $\vec{\mathfrak{v}}$.

Fauzi et al. (2016) assume that the mix network is structured as follows. First, the encrypters (e.g., voters) prove that their ciphertexts (e.g., *encrypted* ballots) are admissible. More precisely, by using a *validity argument*, a voter proves that each component (e.g., an SXDH [16] ciphertext consists of three group elements) of her ciphertext is equal to $\mathfrak{g}_1^{F(\vec{\chi})}$, where the polynomial $F(\vec{X})$ has specific form. The validity argument guarantees that the input ciphertexts to the first mix server have been computed only from certain, "allowed", elements of the CRS.

Each mix server first verifies the validity of original (unshuffled) ciphertexts and the soundness of each previous shuffle argument. After that the mix server produces her shuffle $(\vec{\mathfrak{v}}'_i)_{i=1}^n$ together with her shuffle argument π_{sh}. This means that we consider shuffling a part of the shuffle argument.

The generic approach in the shuffle argument is as follows. First let the prover (a mix server) choose a permutation matrix and then commit separately to its every row. The prover then proves that the committed matrix is a permutation matrix, by proving that each row is 1-sparse (i.e., it has at most one non-zero element) as in [30], while computing the last row explicitly. The 1-sparsity argument is based loosely on Square Span Programs [13]. Basically, to show that a vector $\vec{a}$ is 1-sparse, Fauzi et al. (2016) construct $n+1$ polynomials $(P_i(X))_{i=0}^n$ that interpolate a certain matrix (and a certain vector) connected to the definition of 1-sparsity, and then commit to $\vec{a}$ by using a "polynomial" version of the extended Pedersen commitment scheme, $\mathfrak{c} \leftarrow \mathfrak{g}_2^{\sum a_i P_i(\chi) + r\varrho}$, for random secrets χ and ϱ.

To obtain the full shuffle argument, Fauzi et al. (2016) use the same underlying idea as [26, 30, 17]. Namely, they construct a specific *consistency* verification equation that ensures that $(\vec{\mathfrak{v}}_i)_{i=1}^n$ is permuted to $(\vec{\mathfrak{v}}'_i)_{i=1}^n$ by using the same permutation matrix that was used to permute $(\mathfrak{g}_2^{P_i(\chi)})_{i=1}^n$ to $(\mathfrak{A}_{i2})_{i=1}^n$. This is done by using a pairing equation of type $\prod \hat{e}(\vec{\mathfrak{v}}'_i, \mathfrak{g}_2^{P_i(\chi)}) / \prod \hat{e}(\vec{\mathfrak{v}}_i, \mathfrak{A}_{i2}) = \mathfrak{R}$, where $\mathfrak{R}$ is a value that takes care of the rerandomization (i.e., it depends on the values $\vec{s}$ used to rerandomize $\vec{\mathfrak{v}}$, but not on σ).

Both [26] and [17] had an additional problem here, namely it can be the case that a maliciously created $\vec{\mathfrak{v}}'_i$ depends on $P_j(X)$ (in [26], one has $P_j(X_1, \ldots, X_n) = X_j$, where X_j are independent

random variables) so $\log_{\mathfrak{g}_T} \hat{e}(\vec{\mathfrak{v}}'_i, \mathfrak{g}_2^{P_i(\chi)})$ can depend on $P_j(X)P_i(X)$, for arbitrary i and j. In this case, this equation is not sufficient for soundness, since $\{P_i(X)P_j(X)\}_{i,j\in[1\,..\,n]}$ is not linearly independent (e.g., an adversary can cancel out $P_j(X)P_i(X)$ easily with $-P_i(X)P_j(X)$). Therefore, they had to go through additional complicated steps — that reduced the efficiency of their arguments — to achieve (culpable) soundness even in this case.

Here, such complications are not needed, due to the validity argument. Since the validity argument guarantees that $\vec{\mathfrak{v}}_i$ and $\vec{\mathfrak{v}}'_i$ do not depend on $P_i(X)$, it means that the values $\log_{\mathfrak{g}_T} \hat{e}(\vec{\mathfrak{v}}'_i, \mathfrak{g}_2^{P_i(\chi)})$ and $\log_{\mathfrak{g}_T} \hat{e}(\vec{\mathfrak{v}}_i, \mathfrak{A}_{i2})$ do not depend on $P_i(X)P_j(X)$, which removes the problem evident in both [26] and [17]. On the other hand, [26, 17] solved this problem by proving culpable soundness only, while Fauzi et al. (2016) prove that the new argument satisfies the standard soundness property.

The full GBGM soundness proof of the new shuffle argument is quite intricate. In particular, the verification of the permutation matrix argument results in a system of more than 20 polynomial equations. As some other recent papers like [3, 1], [18] use computer-based tools to solve the latter system. More precisely, they use a computer algebra system to find its Gröbner basis [9], and then continue to find solutions from there on. It is interesting that a simple shuffle argument has such a complicated security proof. On the other hand, both researchers and practitioners can write their own computer algebra code to verify the security proof; this is not possible in many other arguments.

The verification is further optimized by the use of batching techniques [4], thus replacing many pairings with less costly exponentiations. Batching has not been used before in the context of pairing-based shuffle arguments.

2.1.4 Results comparison

Table 2.1 compares the result from [17], [18] and known NIZK shuffle arguments in the CRS model. However, differently from other papers, [26] uses symmetric pairings, and thus its computational and communication complexity is not directly comparable. The prover's computational complexity and the communication includes the computation and sending of the ciphertexts themselves. (This is fair, since different shuffle arguments use different public-key cryptosystems that incur different overhead to these complexity measures.) The highlighted cells in each row are the values with best efficiency, or best security properties.

Finally, each of the CRS-model shuffle arguments relies substantially on the GBGM. The Groth-Lu and Fauzi-Lipmaa shuffles rely on the GBGM to prove security of complicated computational assumptions. The Lipmaa-Zhang shuffle relies on the GBGM to prove security of non-falsifiable knowledge assumptions.

2.2 Preliminaries

Let S_n be the symmetric group on n elements. For a (Laurent) polynomial or a rational function f and its monomial μ, denote by $\mathrm{coeff}_\mu(f)$ the coefficient of μ in f. Write $f(\kappa) \approx_\kappa g(\kappa)$, if $f(\kappa) - g(\kappa)$ is negligible as a function of κ.

2.2.1 Bilinear maps

Let κ be the security parameter. Let q be a prime of length $O(\kappa)$ bits. Assume use of a secure bilinear group generator $\mathsf{genbp}(1^\kappa)$ that returns $\mathsf{gk} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e})$, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are three multiplicative groups of order q, and $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Here, we denote the elements of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ as in $\mathfrak{g}_1$, $\mathfrak{g}_2$, $\mathfrak{g}_T$ (i.e., by using the Fraktur typeface). It is required that $\hat{e}$ is bilinear (i.e., $\hat{e}(\mathfrak{g}_1^a, \mathfrak{g}_2^b) = \hat{e}(\mathfrak{g}_1, \mathfrak{g}_2)^{ab}$), efficiently computable, and non-degenerate. We define $\hat{e}((\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{A}_3), \mathfrak{B}) =$

Table 2.1: A comparison of different NIZK shuffle arguments. We always consider shuffling to be a part of the communication and prover's computation. An n millions clock cycles is considered as a basic unit.

	Groth-Lu	Lipmaa-Zhang	Fauzi-Lipmaa	Fauzi-Lipmaa-Zając
Type of pairings	Symmetric	Asymmetric		
|CRS| in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$	$2n+8$	$(2n+2, 5n+4, 0)$	$(6n+8, 2n+8, 1)$	$(2n+6, n+7, 1)$
Communication	$18n+120$	$(8n+6, 4n+5, 0)$	$(7n+2, 2n, 0)$	$(4n+1, 3n+2, 0)$
		Prover's computation		
Exp. in $(\mathbb{G}_1, \mathbb{G}_2)$	$54n+246$	$(16n+6, 12n+5)$	$(14n+3, 4n)$	$(9n+2, 9n+3)$
Units		36	19.8	24.3
		Verifier's computation		
Exp. in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$	—	—	—	$(11n+5, 3n+6, 1)$
Pairings	$75n+282$	$28n+18$	$18n+6$	$3n+6$
Units		196	126	36.3
Knowl. assumpt-s	No	Yes	Yes	No
Relying on GBGM	PP, SP	Knowledge	Knowl., PSP	Complete
Random oracle		No		
Soundness	Culpable	Full	Culpable	Full

$(\hat{e}(\mathfrak{A}_1, \mathfrak{B}), \hat{e}(\mathfrak{A}_2, \mathfrak{B}), \hat{e}(\mathfrak{A}_3, \mathfrak{B}))$ and $\hat{e}(\mathfrak{B}, (\mathfrak{A}_1, \mathfrak{A}_2, \mathfrak{A}_3)) = (\hat{e}(\mathfrak{B}, \mathfrak{A}_1), \hat{e}(\mathfrak{B}, \mathfrak{A}_2), \hat{e}(\mathfrak{B}, \mathfrak{A}_3))$. Assume that $\mathfrak{g}_i$ is a generator of $\mathbb{G}_i$ for $i \in \{1, 2\}$, and set $\mathfrak{g}_T \leftarrow \hat{e}(\mathfrak{g}_1, \mathfrak{g}_2)$.

For $\kappa = 128$, the current recommendation is to use an optimal (asymmetric) Ate pairing over a subclass of Barreto-Naehrig curves. In that case, at security level of $\kappa = 128$, an element of $\mathbb{G}_1/\mathbb{G}_2/\mathbb{G}_T$ can be represented in respectively 256/512/3072 bits.

2.2.2 Zero knowledge

A NIZK argument for a group-dependent language $\mathcal{L}$ consists of four algorithms, setup, gencrs, pro and ver. The setup algorithm setup takes as input 1^κ and n (the input length), and outputs the group description gk. The CRS generation algorithm gencrs takes as input gk and outputs the prover's CRS crs_p, the verifier's CRS crs_v, and a trapdoor td. The distinction between crs_p and crs_v is only important for efficiency. The prover pro takes as input gk and crs_p, a statement u, and a witness w, and outputs an argument π. The verifier ver takes as input gk and crs_v, a statement u, and an argument π, and either accepts or rejects.

Some of the properties of an argument are: (i) *perfect completeness* (honest verifier always accepts honest prover's argument), (ii) *perfect zero knowledge* (there exists an efficient simulator that can, given u, $(\mathsf{crs}_p, \mathsf{crs}_v)$ and td, output an argument that comes from the same distribution as the argument produced by the prover), (iii) *adaptive computational soundness* (if $u \notin \mathcal{L}$, then an arbitrary non-uniform probabilistic polynomial time prover has negligible probability of success in creating a satisfying argument), and (iv) *adaptive computational culpable soundness* [26, 27] (if $u \notin \mathcal{L}$, then an arbitrary NUPPT prover has negligible success in creating a satisfying argument together with a witness that $u \notin \mathcal{L}$). An argument is an *argument of knowledge*, if from an accepting argument it follows that the prover knows the witness. Below we define these notions formally.

Let $\mathcal{R} = \{(u, w)\}$ be an efficiently computable binary relation with $|w| = \mathrm{poly}(|u|)$. Here, u is a statement, and w is a witness. Let $\mathcal{L} = \{u : \exists w, (u, w) \in \mathcal{R}\}$ be an **NP**-language. Let $n = |u|$ be the input length. For fixed n, we have a relation $\mathcal{R}_n$ and a language $\mathcal{L}_n$. Here, as in [26], since we argue about group elements, both $\mathcal{L}_n$ and $\mathcal{R}_n$ are group-dependent and thus we add gk as an input to $\mathcal{L}_n$ and $\mathcal{R}_n$. Let $\mathcal{R}_n(\mathsf{gk}) := \{(u, w) : (\mathsf{gk}, u, w) \in \mathcal{R}_n\}$.

A *non-interactive argument* for a group-dependent relation family $\mathcal{R}$ consists of four PPT algorithms: a setup algorithm setup, a common reference string (CRS) generator gencrs, a prover pro, and a verifier ver. For $\mathsf{gk} \leftarrow \mathsf{setup}(1^\kappa, n)$ (where n is the input length) and $(\mathsf{crs} = (\mathsf{crs}_p, \mathsf{crs}_v), \mathsf{td}) \leftarrow \mathsf{gencrs}(\mathsf{gk})$ (where td is not accessible to anybody but the simulator), $\mathsf{pro}(\mathsf{crs}_p; u, w)$ produces an argument π, and $\mathsf{ver}(\mathsf{crs}_v; u, \pi)$ outputs either 1 (accept) or 0 (reject). Here, crs_p (resp., crs_v) is the part of the CRS given to the prover (resp., the verifier). Distinction between crs_p and crs_v is not important from the security point of view, but in many cases crs_v is significantly shorter.

A non-interactive argument Ψ is *perfectly complete*, if for all $n = \mathrm{poly}(\kappa)$,

$$\Pr\left[\begin{array}{l}\mathsf{gk} \leftarrow \mathsf{setup}(1^\kappa, n), ((\mathsf{crs}_p, \mathsf{crs}_v), \mathsf{td}) \leftarrow \mathsf{gencrs}(\mathsf{gk}), (u, w) \leftarrow \mathcal{R}_n(\mathsf{gk}) : \\ \mathsf{ver}(\mathsf{gk}, \mathsf{crs}_v; u, \mathsf{pro}(\mathsf{gk}, \mathsf{crs}_p; u, w)) = 1\end{array}\right] = 1 \ .$$

Ψ is adaptively *computationally sound* for $\mathcal{L}$, if for all $n = \mathrm{poly}(\kappa)$ and non-uniform probabilistic polynomial-time adv,

$$\Pr\left[\begin{array}{l}\mathsf{gk} \leftarrow \mathsf{setup}(1^\kappa, n), ((\mathsf{crs}_p, \mathsf{crs}_v), \mathsf{td}) \leftarrow \mathsf{gencrs}(\mathsf{gk}), \\ (u, \pi) \leftarrow \mathsf{adv}(\mathsf{gk}, \mathsf{crs}_p, \mathsf{crs}_v) : (\mathsf{gk}, u) \notin \mathcal{L}_n \wedge \mathsf{ver}(\mathsf{gk}, \mathsf{crs}_v; u, \pi) = 1\end{array}\right] \approx_\kappa 0 \ .$$

We recall that in situations where the inputs have been committed by using a computationally binding trapdoor commitment scheme, the notion of computational soundness does not make sense (since the commitments could be to any input messages). Instead, one should either proof culpable soundness or the argument of knowledge property.

Ψ is adaptively *computationally culpably sound* [26, 27] for $\mathcal{L}$, if for all $n = \mathrm{poly}(\kappa)$, for all polynomial-time decidable binary relations $\mathcal{R}^{\mathsf{guilt}} = \{\mathcal{R}_n^{\mathsf{guilt}}\}$ consisting of elements from $\bar{\mathcal{L}}$ and witnesses w^{guilt}, and for all non-uniform probabilistic polynomial-time adv,

$$\Pr\left[\begin{array}{l}\mathsf{gk} \leftarrow \mathsf{setup}(1^\kappa, n), ((\mathsf{crs}_p, \mathsf{crs}_v), \mathsf{td}) \leftarrow \mathsf{gencrs}(\mathsf{gk}), \\ (u, \pi, w^{\mathsf{guilt}}) \leftarrow \mathsf{adv}(\mathsf{gk}, \mathsf{crs}_p, \mathsf{crs}_v) : (\mathsf{gk}, u, w^{\mathsf{guilt}}) \in \mathcal{R}_n^{\mathsf{guilt}} \wedge \mathsf{ver}(\mathsf{gk}, \mathsf{crs}_v; u, \pi) = 1\end{array}\right] \approx_\kappa 0 \ .$$

For algorithms adv and X_{adv}, we write $(y; y') \leftarrow (\mathsf{adv}||X_{\mathsf{adv}})(\chi)$ if adv on input χ outputs y, and X_{adv} on the same input (including the random tape of adv) outputs y'.

Ψ is *an argument of knowledge*, if for all $n = \mathrm{poly}(\kappa)$ and every non-uniform probabilistic polynomial-time adv, there exists a non-uniform probabilistic polynomial-time extractor X, s.t. for every auxiliary input $\mathsf{aux} \in \{0,1\}^{\mathrm{poly}(\kappa)}$,

$$\Pr\left[\begin{array}{l}\mathsf{gk} \leftarrow \mathsf{setup}(1^\kappa, n), ((\mathsf{crs}_p, \mathsf{crs}_v), \mathsf{td}) \leftarrow \mathsf{gencrs}(\mathsf{gk}), \\ ((u, \pi); w) \leftarrow (\mathsf{adv}||X_{\mathsf{adv}})(\mathsf{crs}_p, \mathsf{crs}_v; \mathsf{aux}) : (u, w) \notin \mathcal{R} \wedge \mathsf{ver}(\mathsf{crs}_v; u, \pi) = 1\end{array}\right] \approx_\kappa 0 \ .$$

Here, aux can be seen as the common auxiliary input to adv and X_{adv} that is generated by using benign auxiliary input generation [5].

Ψ is *perfectly zero-knowledge*, if there exists a probabilistic polynomial-time simulator $\mathcal{X}_\gamma$, such that for all stateful non-uniform probabilistic adversaries adv and $n = \mathrm{poly}(\kappa)$,

$$\Pr\left[\begin{array}{l}\mathsf{gk} \leftarrow \mathsf{setup}(1^\kappa, n), \\ ((\mathsf{crs}_p, \mathsf{crs}_v), \mathsf{td}) \leftarrow \mathsf{gencrs}(\mathsf{gk}), \\ (u, w) \leftarrow \mathsf{adv}(\mathsf{gk}, \mathsf{crs}_p, \mathsf{crs}_v), \\ \pi \leftarrow \mathsf{pro}(\mathsf{gk}, \mathsf{crs}_p; u, w) : \\ (\mathsf{gk}, u, w) \in \mathcal{R}_n \wedge \mathsf{adv}(\mathsf{gk}, \pi) = 1\end{array}\right] = \Pr\left[\begin{array}{l}\mathsf{gk} \leftarrow \mathsf{setup}(1^\kappa, n), \\ ((\mathsf{crs}_p, \mathsf{crs}_v); \mathsf{td}) \leftarrow \mathsf{gencrs}(\mathsf{gk}), \\ (u, w) \leftarrow \mathsf{adv}(\mathsf{gk}, \mathsf{crs}_p, \mathsf{crs}_v), \\ \pi \leftarrow \mathcal{X}_\gamma(\mathsf{gk}, \mathsf{crs}_p, \mathsf{crs}_v; u, \mathsf{td}) : \\ (\mathsf{gk}, u, w) \in \mathcal{R}_n \wedge \mathsf{adv}(\mathsf{gk}, \pi) = 1\end{array}\right]$$

Here, the prover and the simulator use the same CRS. It is *same-string zero knowledge*. A same-string statistical zero knowledge argument stays secure even when using the CRS an unbounded number of times.

2.2.3 Generic bilinear group model

The soundness of [11] and new assumption in [17] are proven in the generic bilinear group model (GBGM, [36, 31, 8]). Here we present this model of computation using description based on [31].

We start by picking a random asymmetric bilinear group $\mathsf{gk} := (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \leftarrow \mathsf{genbp}(1^\kappa)$. Consider a black box $\mathbf{B}$ that can store values from groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ in internal state variables $\mathsf{cell}_1, \mathsf{cell}_2, \ldots$, where for simplicitly we allow the storage space to be infinite (this only increases the power of a generic adversary). The initial state consists of some values $(\mathsf{cell}_1, \mathsf{cell}_2, \ldots, \mathsf{cell}_{|inp|})$, which are set according to some probability distribution. Each state variable cell_i has an accompanying type $\mathsf{type}_i \in \{1, 2, T, \bot\}$. We assume initially $\mathsf{type}_i = \bot$ for $i > |inp|$. The black box allows computation operations on internal state variables and queries about the internal state. No other interaction with $\mathbf{B}$ is possible.

Let Π be the allowed set of computation operations. A computation operation consists of selecting a (say, t-ary) operation $f \in \Pi$ together with $t+1$ indices $i_1, i_2, \ldots, i_{t+1}$. Assuming inputs have the correct type, $\mathbf{B}$ computes $f(\mathsf{cell}_{i_1}, \ldots, \mathsf{cell}_{i_t})$ and stores the result in $\mathsf{cell}_{i_{t+1}}$. For a set Σ of relations, a query consists of selecting a (say, t-ary) relation $\varrho \in \Sigma$ together with t indices $i_1, i_2, \ldots, i_t$. Assuming inputs have the correct type, $\mathbf{B}$ replies to the query with $\varrho(\mathsf{cell}_{i_1}, \ldots, \mathsf{cell}_{i_t})$.

In the GBGM, we define $\Pi = \{\cdot, \hat{e}\}$ and $\Sigma = \{=\}$, where

1. On input $(\cdot, i_1, i_2, i_3)$: if $\mathsf{type}_{i_1} = \mathsf{type}_{i_2} \neq \bot$ then set $\mathsf{cell}_{i_3} \leftarrow \mathsf{cell}_{i_1} \cdot \mathsf{cell}_{i_2}$ and $\mathsf{type}_{i_3} \leftarrow \mathsf{type}_{i_1}$.
2. On input $(\hat{e}, i_1, i_2, i_3)$: if $\mathsf{type}_{i_1} = 1$ and $\mathsf{type}_{i_2} = 2$ then set $\mathsf{cell}_{i_3} \leftarrow \hat{e}(\mathsf{cell}_{i_1}, \mathsf{cell}_{i_2})$ and $\mathsf{type}_{i_3} \leftarrow T$.
3. On input $(=, i_1, i_2)$: if $\mathsf{type}_{i_1} = \mathsf{type}_{i_2} \neq \bot$ and $\mathsf{cell}_{i_1} = \mathsf{cell}_{i_2}$ then return 1. Otherwise return 0.

Since we are proving lower bounds, we will give a generic adversary adv additional power. We assume that all relation queries are for free. We also assume that adv is successful if after τ operation queries, he makes an equality query $(=, i_1, i_2)$, $i_1 \neq i_2$, that returns 1; at this point adv quits. Thus, if $\mathsf{type}_i \neq \bot$, then $\mathsf{cell}_i = F_1(\mathsf{cell}_1, \ldots, \mathsf{cell}_{|inp|})$ for a polynomial F_i known to adv.

The GBGM has proved itself to be very fruitful since its introduction, [8]. In particular, the generic (bilinear) group model is amenable to computerized analysis, and as such, has proven itself to be very useful say in the area of structure-preserving signature schemes [3]; see also [1].

Finally, Fischlin [19] and Dent [14] have pointed out that there exist constructions that are secure in (Shoup's version of) the generic group model but cannot be instantiated given any efficient instantiation of the group encoding. However, their constructions are utterly artificial; e.g., Dent constructed a signature scheme that under certain conditions outputs the signing key as a part of the signature.

2.3 FL shuffle argument

In this section we describe subarguments that constitute the shuffle argument from [17].

2.3.1 Unit vector argument

In a unit vector argument, the prover aims to convince the verifier that he knows how to open a commitment $(\mathfrak{A}_1, \mathfrak{A}_2^\gamma)$ to *some* $(\vec{e}_I, r)$, where $\vec{e}_I$ denotes the Ith unit vector for $I \in [1\,..\,n]$. Fauzi et al. (2016) construct the unit vector argument by using square span programs (SSP-s, [13], an especially efficient variant of the quadratic arithmetic programs of [20]).

Clearly, $\vec{a} \in \mathbb{Z}_p^n$ is a unit vector iff the following $n+1$ conditions hold:

- $a_i \in \{0, 1\}$ for $i \in [1\,..\,n]$ (i.e., $\vec{a}$ is Boolean), and

- $\sum_{i=1}^{n} a_i = 1$.

Following the methodology of [13], [17] obtains an efficient NIZK argument out of these conditions. Let $\{0,2\}^{n+1}$ denote the set of $(n+1)$-dimensional vectors where every coefficient is from $\{0,2\}$, let $\circ$ denote the Hadamard (entry-wise) product of two vectors, let $V := \left(\begin{smallmatrix} 2\cdot I_{n\times n} \\ \vec{1}_n^\top \end{smallmatrix}\right) \in \mathbb{Z}_p^{(n+1)\times n}$ and $\vec{b} := \left(\begin{smallmatrix} \vec{0}_n \\ 1 \end{smallmatrix}\right) \in \mathbb{Z}_p^{n+1}$. Clearly, the above $n+1$ conditions hold iff $V\vec{a} + \vec{b} \in \{0,2\}^{n+1}$, i.e.,

$$(V\vec{a} + \vec{b} - \vec{1}_{n+1}) \circ (V\vec{a} + \vec{b} - \vec{1}_{n+1}) = \vec{1}_{n+1} \ . \tag{2.1}$$

Let ω_i, $i \in [1\,..\,n+1]$ be $n+1$ different values. Let $Z(X) := \prod_{i=1}^{n+1}(X - \omega_i)$ be the unique degree $n+1$ monic polynomial, such that $Z(\omega_i) = 0$ for all $i \in [1\,..\,n+1]$. Let the ith Lagrange basis polynomial $\ell_i(X) := \prod_{i,j\in[1\,..\,n+1],j\neq i}((X-\omega_j)/(\omega_i-\omega_j))$ be the unique degree n polynomial, s.t. $\ell_i(\omega_i) = 1$ and $\ell_i(\omega_j) = 0$ for $j \neq i$. For a vector $\vec{x} \in \mathbb{Z}_p^{n+1}$, let $L_{\vec{x}}(X) = \sum_{i=1}^{n+1} x_i\ell_i(X)$ be a degree n polynomial that interpolates $\vec{x}$, i.e., $L_{\vec{x}}(\omega_i) = x_i$.

For $i \in [1\,..\,n]$, let $y_i(X)$ be the polynomial that interpolates the ith column of the matrix V. That is, $y_i(X) = 2\ell_i(X) + \ell_{n+1}(X)$ for $i \in [1\,..\,n]$. Let $y_0(X) = -1 + \ell_{n+1}(X)$ be the polynomial that interpolates $\vec{b} - \vec{1}_{n+1}$. The polynomial commitment scheme is instantiated with $\mathcal{F}_{\mathsf{com}} = (Z(X), (y_i(X))_{i=1}^n)$.

As in [13], Fauzi et al. (2016) arrive at the polynomial $Q(X) = (\sum_{i=1}^n a_i y_i(X) + y_0(X))^2 - 1 = (y_I(X) + y_0(X))^2 - 1$ (here, we used the fact that $\vec{a} = \vec{e_I}$ for some $I \in [1\,..\,n]$), such that $\vec{a}$ is a unit vector iff $Z(X) \mid Q(X)$. As in [20, 13], to obtain privacy, randomness to $Q(X)$ is added, arriving at the degree $2(n+1)$ polynomial $Q_{wi}(X) = (rZ(X) + y_I(X) + y_0(X))^2 - 1$. By [20, 13], Eq. (2.1) holds iff

(i) $Q_{wi}(X) = (A(X) + y_0(X))^2 - 1$, where $A(X) = r_a Z(X) + \sum_{i=1}^n a_i y_i(X) \in \mathrm{span}(\mathcal{F}_{\mathsf{com}})$, and
(ii) $Z(X) \mid Q_{wi}(X)$.

An honest prover computes the degree $\leq n+1$ polynomial $\pi_{wi}(X) \leftarrow Q_{wi}(X)/Z(X) \in \mathbb{Z}_p[X]$, and sets the argument to be equal to $\pi_{uv}^* := \mathfrak{g}_1^{\pi_{wi}(\chi)}$ for a secret χ that instantiates X. If it exists, $\pi_{wi}(X) := Q_{wi}(X)/Z(X)$ is equal to $r^2Z(X) + r\cdot 2(y_I(X) + y_0(X)) + \Pi_I(X)$, where for $i \in [1\,..\,n]$, $\Pi_i(X) := ((y_i(X) + y_0(X))^2 - 1)/Z(X)$ is a degree $\leq n-1$ polynomial and $Z(X) \mid ((y_i(X) + y_0(X))^2 - 1)$. Thus, computing π_{uv}^* uses two exponentiations.

Fauzi et al. (2016) use a knowledge (PKE) assumption in a standard way to guarantee that $A(X)$ is in the span of $\{X^i\}_{i=0}^{n+1}$. As in [20, 13], they then guarantee condition (i) by using a PCDH assumption and condition (ii) by using a TSDH assumption. Here, [17] uses the same technique as in [20] and subsequent papers by introducing an additional secret, β, and adding one group element $\mathfrak{A}_1^\beta$ to the argument.

2.3.2 New same-message argument

In a *same-message argument*, the prover aims to convince the verifier that he knows, given two commitment keys ck and $\widehat{\mathsf{ck}}$ (that correspond to two tuples of polynomials $(P_i(X))_{i=0}^n$ and $(\hat{P}_i(X))_{i=0}^n$, respectively), how to open $(\mathfrak{A}_1, \mathfrak{A}_2^\gamma) = \mathsf{com}(\mathsf{ck}; \vec{m}; r)$ and $(\hat{\mathfrak{A}}_1, \hat{\mathfrak{A}}_2^{\hat{\gamma}}) = \mathsf{com}(\widehat{\mathsf{ck}}; \vec{m}; \hat{r})$ as commitments (w.r.t. ck and $\widehat{\mathsf{ck}}$) to the same plaintext vector $\vec{m}$ (but not necessarily to the same randomizer r).

Fauzi et al. (2016) propose an efficient same-message argument using $\mathcal{F}_{\mathsf{com}} = (Z(X), (y_i(X))_{i=1}^n)$ as described in Sect. 2.3.1. In the shuffle argument, they need $(\hat{P}_i(X))_{i=0}^n$ to satisfy some specific requirements w.r.t. $\mathcal{F}_{\mathsf{com}}$, see Sect. 2.3.3. The choice of $\hat{P}_i$ is free otherwise.

Denote $\hat{Z}(X) = \hat{P}_0(X)$. For the same-message argument to be an argument of knowledge *and* efficient, [17] chooses $\hat{P}_i$ such that $(\hat{P}_i(\omega_j))_{j=1}^{n+1} = (y_i(\omega_j))_{j=1}^{n+1} = 2\vec{e_i} + \vec{e_{n+1}}$ for $i \in [1\,..\,n]$. Moreover, $(\hat{Z}(\omega_j))_{j=1}^{n+1} = (Z(\omega_j))_{j=1}^{n+1} = \vec{0}_{n+1}$.

Following similar methodology as in Sect. 2.3.1, define

$$Q_{wi}(X) := (\hat{r}\hat{Z}(X) + \sum_{i=1}^{n} \hat{m}_i \hat{P}_i(X)) - (rZ(X) + \sum_{i=1}^{n} m_i y_i(X)) \ .$$

Let $\hat{n}$ be the maximum degree of polynomials in $(y_i(X), \hat{P}_i(X))_{i=0}^{n}$, thus $\deg Q_{wi} \leq \hat{n}$. Since $Q_{wi}(\omega_j) = 2(\hat{m}_j - m_j)$ for $j \in [1\,..\,n]$, $Q_{wi}(\omega_j) = 0$ iff $m_j = \hat{m}_j$. Moreover, if $\vec{m} = \vec{\hat{m}}$ then $Q_{wi}(\omega_{n+1}) = \sum_{i=1}^{n} \hat{m}_i - \sum_{i=1}^{n} m_i = 0$. Hence, $\vec{m} = \vec{\hat{m}}$ iff

(i) $Q_{wi}(X) = \hat{A}(X) - A(X)$, where $A(X) \in \mathrm{span}(\{Z(X)\} \cup \{y_i(X)\}_{i=1}^{n})$, and $\hat{A}(X) \in \mathrm{span}(\{\hat{Z}(X)\} \cup \{\hat{P}_i(X)\}_{i=1}^{n})$, and

(ii) there exists a degree $\leq \hat{n} - (n+1)$ polynomial $\pi_{wi}(X) = Q_{wi}(X)/Z(X)$.

If the prover is honest, then $\pi_{wi}(X) = \hat{r}\hat{Z}(X)/Z(X) - r + \sum m_i \cdot ((\hat{P}_i(X) - y_i(X))/Z(X))$. Note that [17] does not need that $Q_{wi}(X) = 0$ as a polynomial, we just need that $Q_{wi}(\omega_i) = 0$, which is a deviation from the strategy usually used in QAP/QSP-based arguments [20].

2.3.3 New assumption: PSP

We will next describe a new computational assumption (PSP) that is needed in the shuffle argument proposed by Fauzi et al. (2016). The PSP assumption is related to but not equal to the SP assumption from [26]. Interestingly, the generic group proof of the PSP assumption relies on the Schwartz-Zippel lemma, while in most of the known interactive shuffle arguments (like [32]), the Schwartz-Zippel lemma is used in the reduction from the shuffle security to some underlying assumption.

Let let $d(n) > n$ be a function. Let $\hat{\mathcal{F}} = (\hat{P}_i(X))_{i=0}^{n}$ be a tuple of polynomials. We say $(d(n), \hat{\mathcal{F}})$ is *PSP-friendly*, if the following set is linearly independent: $\hat{\mathcal{F}}_{d(n)} := \{X^i\}_{i=0}^{2d(n)} \cup \{X^i \cdot \hat{P}_j(X)\}_{0 \leq i \leq d(n), 0 \leq j \leq n} \cup \{\hat{P}_0(X)\hat{P}_j(X)\}_{j=0}^{n}$.

Let $(d(n), \hat{\mathcal{F}})$ be PSP-friendly. Let $\mathcal{F} = (P_i(X))_{i=0}^{n}$ be a tuple of polynomials of degree $\leq d(n)$. The $(\mathcal{F}, \hat{\mathcal{F}})$-*Power Simultaneous Product (PSP) assumption* states that for any $n = \mathrm{poly}(\kappa)$ and any NUPPT adversary adv,

$$\Pr\left[\begin{array}{l} \mathsf{gk} \leftarrow \mathsf{genbp}(1^\kappa, n), (\mathfrak{g}_1, \mathfrak{g}_2, \chi) \leftarrow_R \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{Z}_p, \\ \mathbb{G}_1^{n+2} \ni (\mathfrak{t}, \hat{\mathfrak{t}}, (\mathfrak{s}_i)_{i=1}^{n}) \leftarrow \mathsf{adv}(\mathsf{gk}; ((\mathfrak{g}_1, \mathfrak{g}_2)^{\chi^i})_{i=0}^{d(n)}, (\mathfrak{g}_1, \mathfrak{g}_2)^{\hat{\mathcal{F}}(\chi)}) : \\ \mathfrak{t}^{P_0(\chi)} \cdot \prod_{i=1}^{n} \mathfrak{s}_i^{P_i(\chi)} = \hat{\mathfrak{t}}^{\hat{P}_0(\chi)} \cdot \prod_{i=1}^{n} \mathfrak{s}_i^{\hat{P}_i(\chi)} = 1 \wedge (\exists i \in [1\,..\,n] : \mathfrak{s}_i \neq 1) \end{array}\right] \approx_\kappa 0 \ .$$

PSP-friendliness and the PSP assumption are defined so that both the generic model proof and the reduction from the shuffle soundness to the PSP in Thm. 2 would go through. As in the case of SP, it is essential that two simultaneous products have to hold true; the simpler version of the PSP assumption with only one product (i.e., $\mathfrak{t}^{P_0(\chi)} \cdot \prod_{i=1}^{n} \mathfrak{s}_i^{P_i(\chi)} = 1$) does not hold in the generic bilinear group model. Differently from SP, the PSP assumption incorporates possibly distinct $\mathfrak{t}$ and $\hat{\mathfrak{t}}$ since the same-message argument does not guarantee that the randomizers of two commitments are equal.

2.3.4 Shuffle argument

Let ElGamal operate in $\mathbb{G}_1$ defined by gk. In a shuffle argument, the prover aims to convince the verifier that, given the description of a group, a public key, and two vectors of ciphertexts, the second vector of the ciphertexts is a permutation of rerandomized versions of the ciphertexts from

the first vector. However, to achieve better efficiency, [17] constructs a shuffle argument that is only culpably sound with respect to the next relation (i.e., $\mathcal{R}_{sh}^{\mathsf{guilt}}$-sound):

$$\mathcal{R}_{sh,n}^{\mathsf{guilt}} = \left\{ \begin{array}{l} (\mathsf{gk}, (\mathsf{pk}, (\mathfrak{v}_i)_{i=1}^n, (\mathfrak{v}'_i)_{i=1}^n), \mathsf{sk}) : \mathsf{gk} \in \mathsf{genbp}(1^\kappa, n) \wedge \\ (\mathsf{pk}, \mathsf{sk}) \in \mathsf{genpkc}(\mathsf{gk}) \wedge \left(\forall \sigma \in S_n : \exists i : \mathsf{dec}_{\mathsf{sk}}(\mathfrak{v}'_i) \neq \mathsf{dec}_{\mathsf{sk}}(\mathfrak{v}_{\sigma(i)})\right) \end{array} \right\} .$$

The argument of [26] is proven to be $\mathcal{R}_{sh}^{\mathsf{guilt}}$-sound with respect to the same relation. See [26] or the introduction for an explanation why $\mathcal{R}_{sh}^{\mathsf{guilt}}$ is sufficient.

As noted in the introduction, [17] needs to use same-message arguments and rely on the PSP assumption. Thus, they need polynomials $\hat{P}_j$ that satisfy two different requirements at once. First, to be able to use the same-message argument, they need that $y_j(\omega_k) = \hat{P}_j(\omega_k)$ for $k \in [1\,..\,n+1]$. Second, to be able to use the PSP assumption, they need $(d, \hat{\mathcal{F}})$ to be PSP-friendly, and for this they need $\hat{P}_j(X)$ to have a sufficiently large degree. Recall that y_j are fixed by the unit vector argument. In the paper, the authors show that such a choice for $\hat{P}_j$ exists.

Proposition 1. *Let* $\hat{y}_j(X) := (XZ(X)+1)^{j-1}(X^2Z(X)+1)y_j(X)$ *for* $j \in [1\,..\,n]$, *and* $\hat{Z}(X) = \hat{y}_0(X) := (XZ(X)+1)^{n+1}Z(X)$. *Let* $\hat{\mathcal{F}}_{\mathsf{com}} = (\hat{y}_j(X))_{j=0}^n$. *Then* $\hat{y}_j(\omega_k) = y_j(\omega_k)$ *for all* j, k, *and* $(n+1, \hat{\mathcal{F}}_{\mathsf{com}})$ *is PSP-friendly.*

Next, we will provide the full description of the [17] shuffle argument. Note that $(\mathfrak{c}_i)_{i=1}^n$ are commitments to the rows of the permutation matrix $\vec{A}$, proven by the n unit vector arguments $(\pi_{uv,i})_{i=1}^n$ and by the implicit computation of $\mathfrak{c}_n$. We denote $\hat{e}((\mathfrak{a}, \mathfrak{b}), \mathfrak{c}) := (\hat{e}(\mathfrak{a}, \mathfrak{c}), \hat{e}(\mathfrak{b}, \mathfrak{c}))$.

System parameters: Let $(\mathsf{genpkc}, \mathsf{enc}, \mathsf{dec})$ be the ElGamal cryptosystem. Let com be the polynomial commitment scheme. Consider polynomials $\mathcal{F}_{\mathsf{com}} = \{Z(X)\} \cup (y_i(X))_{i=1}^n$ from Sect. 2.3.1. Let $\hat{\mathcal{F}}_{\mathsf{com}} = (\hat{y}_i(X))_{i=0}^n$ be as in Prop. 1.

Setup $\mathsf{setup}_{sh}(1^\kappa, n)$**:** Let $\mathsf{gk} \leftarrow \mathsf{genbp}(1^\kappa, n)$.

CRS generation $\mathsf{gencrs}_{sh}(\mathsf{gk})$**:** Let $(\mathfrak{g}_1, \mathfrak{g}_2, \chi, \beta, \gamma) \leftarrow_r \mathbb{G}_1^* \times \mathbb{G}_2^* \times \mathbb{Z}_p^3$ with $Z(\chi) \neq 0$. Let $(\mathsf{crs}_{uv,p}, \mathsf{crs}_{uv,v}) \leftarrow_r \mathsf{gencrs}_{uv}(\mathsf{gk}, n)$, $(\mathsf{crs}_{sm,p}, \mathsf{crs}_{sm,v}) \leftarrow_r \mathsf{gencrs}_{sm}(\mathsf{gk}, n)$, but by using the same $(\mathfrak{g}_1, \mathfrak{g}_2, \chi, \beta, \gamma)$ in both cases. Let $\mathsf{ck} \leftarrow (\mathfrak{g}_1, \mathfrak{g}_2^\gamma)^{\mathcal{F}_{\mathsf{com}}(\chi)}$ and $\widehat{\mathsf{ck}} \leftarrow (\mathfrak{g}_1, \mathfrak{g}_2^{\hat{\gamma}})^{\hat{\mathcal{F}}_{\mathsf{com}}(\chi)}$. Set $(\mathfrak{D}_1, \mathfrak{D}_2^\gamma) \leftarrow \mathsf{com}(\mathsf{ck}; \vec{1_n}; 0)$, $(\hat{\mathfrak{D}}_1, \hat{\mathfrak{D}}_2^{\hat{\gamma}}) \leftarrow \mathsf{com}(\widehat{\mathsf{ck}}; \vec{1_n}; 0)$. Set $\mathsf{crs}_{sh,p} \leftarrow (\mathsf{crs}_{uv,p}, \widehat{\mathsf{ck}}, \mathfrak{g}_1^{\hat{Z}(\chi)/Z(\chi)}, \mathfrak{g}_1, (\mathfrak{g}_1^{(\hat{y}_i(\chi)-y_i(\chi))/Z(\chi)})_{i=1}^n, \mathfrak{D}_1, \mathfrak{D}_2^\gamma, \hat{\mathfrak{D}}_1, \hat{\mathfrak{D}}_2^{\hat{\gamma}})$, $\mathsf{crs}_{sh,v} \leftarrow (\mathsf{crs}_{uv,v}, \mathfrak{g}_2^{\hat{\gamma}}, \{\mathfrak{g}_2^{\gamma y_i(\chi)}, \mathfrak{g}_2^{\hat{\gamma}\hat{y}_i(\chi)}\}_{i=0}^n, \mathfrak{D}_1, \mathfrak{D}_2^\gamma, \hat{\mathfrak{D}}_1, \hat{\mathfrak{D}}_2^{\hat{\gamma}})$, and $\mathsf{td}_{sh} \leftarrow \chi$. Return $((\mathsf{crs}_{sh,p}, \mathsf{crs}_{sh,v}), \mathsf{td}_{sh})$.

Common input: $(\mathsf{pk}, (\mathfrak{v}_i, \mathfrak{v}'_i)_{i=1}^n)$, where $\mathsf{pk} = (\mathfrak{g}_1, \mathfrak{h}) \in \mathbb{G}_1^2$, $\mathfrak{v}_i \in \mathbb{G}_1^2$ and $\mathfrak{v}'_i = \mathfrak{v}_{\sigma(i)} \cdot \mathsf{enc}_{\mathsf{pk}}(1; t_i) \in \mathbb{G}_1^2$.

Argument $\mathsf{pro}_{sh}(\mathsf{gk}, \mathsf{crs}_{sh,p}; \mathsf{pk}, (\mathfrak{v}_i, \mathfrak{v}'_i)_{i=1}^n; \sigma, (t_i)_{i=1}^n)$**:**

(1) Let $\vec{A} = \vec{A}_{\sigma^{-1}}$ be the $n \times n$ permutation matrix corresponding to σ^{-1}.
(2) For $i \in [1\,..\,n-1]$:
   - Set $r_i \leftarrow \mathbb{Z}_p$, $(\mathfrak{c}_{i1}, \mathfrak{c}_{i2}^\gamma) \leftarrow \mathsf{com}(\mathsf{ck}; \vec{A}_i; r_i)$, $(\hat{\mathfrak{c}}_{i1}, \hat{\mathfrak{c}}_{i2}^{\hat{\gamma}}) \leftarrow \mathsf{com}(\widehat{\mathsf{ck}}; \vec{A}_i; r_i)$.

(3) Set $r_n \leftarrow -\sum_{i=1}^{n-1} r_i$, $(\mathfrak{c}_{n1}, \mathfrak{c}_{n2}^\gamma) \leftarrow (\mathfrak{D}_1, \mathfrak{D}_2^\gamma) / \prod_{i=1}^{n-1} (\mathfrak{c}_{i1}, \mathfrak{c}_{i2}^\gamma)$.
(4) Set $(\hat{\mathfrak{c}}_{n1}, \hat{\mathfrak{c}}_{n2}^{\hat{\gamma}}) \leftarrow (\hat{\mathfrak{D}}_1, \hat{\mathfrak{D}}_2^{\hat{\gamma}}) / \prod_{i=1}^{n-1} (\hat{\mathfrak{c}}_{i1}, \hat{\mathfrak{c}}_{i2}^{\hat{\gamma}})$.
(5) For $i \in [1\,..\,n]$: set $\pi_{uv,i} = (\pi_{uv,i}^*, \mathfrak{c}_{i1}^\beta) \leftarrow \mathsf{pro}_{uv}(\mathsf{gk}, \mathsf{crs}_{uv,p}; \mathfrak{c}_{i1}, \mathfrak{c}_{i2}^\gamma; \vec{A}_i, r_i)$.
(6) Set $r_t \leftarrow_r \mathbb{Z}_p$, $(\mathfrak{d}_1, \mathfrak{d}_2^\gamma) \leftarrow \mathsf{com}(\mathsf{ck}; \vec{t}; r_t)$, and $(\hat{\mathfrak{d}}_1, \hat{\mathfrak{d}}_2^{\hat{\gamma}}) \leftarrow \mathsf{com}(\widehat{\mathsf{ck}}; \vec{t}; r_t)$.
(7) For $i \in [1\,..\,n-1]$:
   - Set $(\pi_{sm,i}^*, \mathfrak{c}_{i1}^\beta) \leftarrow \mathsf{pro}_{sm}(\mathsf{gk}, \mathsf{crs}_{sm,p}; \mathfrak{c}_{i1}, \mathfrak{c}_{i2}^\gamma, \hat{\mathfrak{c}}_{i1}, \hat{\mathfrak{c}}_{i2}^{\hat{\gamma}}; \vec{A}_i, r_i, r_i)$.

(8) Set $\pi_{sm,d} \leftarrow \mathsf{pro}_{sm}(\mathsf{gk}, \mathsf{crs}_{sm,p}; \mathfrak{d}_1, \mathfrak{d}_2^\gamma, \hat{\mathfrak{d}}_1, \hat{\mathfrak{d}}_2^{\hat{\gamma}}; \vec{t}, r_t, r_t)$.
(9) Compute $\mathfrak{U} = (\mathfrak{U}_1, \mathfrak{U}_2) \leftarrow \mathsf{pk}^{r_t} \cdot \prod_{i=1}^n \mathfrak{v}_i^{r_i} \in \mathbb{G}_1^2$. `// The only online step`
(10) Output $\pi_{sh} \leftarrow ((\mathfrak{c}_{i1}, \mathfrak{c}_{i2}^\gamma, \hat{\mathfrak{c}}_{i1}, \hat{\mathfrak{c}}_{i2}^{\hat{\gamma}})_{i=1}^{n-1}, \mathfrak{d}_1, \mathfrak{d}_2^\gamma, \hat{\mathfrak{d}}_1, \hat{\mathfrak{d}}_2^{\hat{\gamma}}, (\pi_{uv,i})_{i=1}^n, (\pi_{sm,i}^*)_{i=1}^{n-1}, \pi_{sm,d}, \mathfrak{U})$

Verification $\mathsf{ver}_{sh}(\mathsf{gk}, \mathsf{crs}_{sh,v}; \mathsf{pk}, (\mathfrak{v}_i, \mathfrak{v}'_i)_{i=1}^n, \pi_{sh})$**:**

(1) Let $(\mathfrak{c}_{n1}, \mathfrak{c}_{n2}^{\gamma}) \leftarrow (\mathfrak{D}_1, \mathfrak{D}_2^{\gamma}) / \prod_{i=1}^{n-1} (\mathfrak{c}_{i1}, \mathfrak{c}_{i2}^{\gamma})$.

(2) Let $(\hat{\mathfrak{c}}_{n1}, \hat{\mathfrak{c}}_{n2}^{\hat{\gamma}}) \leftarrow (\hat{\mathfrak{D}}_1, \hat{\mathfrak{D}}_2^{\hat{\gamma}}) / \prod_{i=1}^{n-1} (\hat{\mathfrak{c}}_{i1}, \hat{\mathfrak{c}}_{i2}^{\hat{\gamma}})$.

(3) For $i \in [1 \,..\, n]$: reject if $\mathsf{ver}_{uv}(\mathsf{gk}, \mathsf{crs}_{uv,v}; \mathfrak{c}_{i1}, \mathfrak{c}_{i2}^{\gamma}; \pi_{uv,i})$ rejects.

(4) For $i \in [1 \,..\, n-1]$: reject if $\mathsf{ver}_{sm}(\mathsf{gk}; \mathsf{crs}_{sm,v}; \mathfrak{c}_{i1}, \mathfrak{c}_{i2}^{\gamma}, \hat{\mathfrak{c}}_{i1}, \hat{\mathfrak{c}}_{i2}^{\hat{\gamma}}; \pi_{sm,i})$ rejects.

(5) Reject if $\mathsf{ver}_{sm}(\mathsf{gk}, \mathsf{crs}_{sm,v}; \mathfrak{d}_1, \mathfrak{d}_2^{\gamma}, \hat{\mathfrak{d}}_1, \hat{\mathfrak{d}}_2^{\hat{\gamma}}; \pi_{sm,d})$ rejects.

(6) Check the PSP-related verification equations: `// The only online step`

(a) $\prod_{i=1}^n \hat{e}(\mathfrak{v}'_i, \mathfrak{g}_2^{\gamma y_i(\chi)}) / \prod_{i=1}^n \hat{e}(\mathfrak{v}_i, \mathfrak{c}_{i2}^{\gamma}) = \hat{e}((\mathfrak{g}_1, \mathfrak{h}), \mathfrak{d}_2^{\gamma}) / \hat{e}(\mathfrak{U}, \mathfrak{g}_2^{\gamma Z(\chi)})$,

(b) $\prod_{i=1}^n \hat{e}(\mathfrak{v}'_i, \mathfrak{g}_2^{\hat{\gamma}\hat{y}_i(\chi)}) / \prod_{i=1}^n \hat{e}(\mathfrak{v}_i, \hat{\mathfrak{c}}_{i2}^{\hat{\gamma}}) = \hat{e}((\mathfrak{g}_1, \mathfrak{h}), \hat{\mathfrak{d}}_2^{\hat{\gamma}}) / \hat{e}(\mathfrak{U}, \mathfrak{g}_2^{\hat{\gamma}\hat{Z}(\chi)})$.

Since $\mathsf{ck}, \widehat{\mathsf{ck}} \subset \mathsf{crs}_{sh,p}$, $(\mathfrak{D}_1, \mathfrak{D}_2^{\gamma}) = \mathsf{com}(\mathsf{ck}; \vec{1}_n; 0)$ and $(\hat{\mathfrak{D}}_1, \hat{\mathfrak{D}}_2^{\hat{\gamma}}) = \mathsf{com}(\widehat{\mathsf{ck}}; \vec{1}_n; 0)$ can be computed from the rest of the CRS. (These four elements are only needed to optimize the computation of $(\mathfrak{c}_{n1}, \mathfrak{c}_{n2}^{\gamma})$ and $(\hat{\mathfrak{c}}_{n1}, \hat{\mathfrak{c}}_{n2}^{\hat{\gamma}})$.) For security, it suffices to take $\mathsf{crs}_{sh}^* = (\mathfrak{g}_1^{\mathcal{F}_{sh,1}(\chi,\beta)}, \mathfrak{g}_2^{\mathcal{F}_{sh,2}(\chi,\beta,\gamma,\hat{\gamma})})$, where $\mathcal{F}_{sh,1} = \mathcal{F}_{uv,1} \cup \hat{\mathcal{F}}_{\mathsf{com}} \cup \{\hat{Z}(X)/Z(X)\} \cup \{(\hat{y}_i(X) - y_i(X))/Z(X)\}_{i=1}^n$ and $\mathcal{F}_{sh,2} = \mathcal{F}_{uv,2} \cup \hat{Y} \cdot (\{1\} \cup \hat{\mathcal{F}}_{\mathsf{com}})$.

Theorem 2. *The new shuffle argument is a non-interactive perfectly complete and perfectly zero-knowledge shuffle argument for ElGamal ciphertexts. If the* $(n+1)$*-TSDH,* $(\hat{n}, \hat{n}+n+2)$*-PCDH,* $(\mathcal{F}_{\mathsf{com}}, \hat{\mathcal{F}}_{\mathsf{com}})$*-PSP,* $(n+1, \mathcal{F}_{sh,1} \setminus (\{1\} \cup \mathcal{F}_{\mathsf{com}}), \mathcal{F}_{sh,2} \setminus Y \cdot (\{1\} \cup \mathcal{F}_{\mathsf{com}}), \gamma)$*-PKE,* $(\hat{\mathcal{F}}_{\mathsf{com}}, \mathcal{F}_{sh,1} \setminus \hat{\mathcal{F}}_{\mathsf{com}}, \mathcal{F}_{sh,2} \setminus \hat{Y}\hat{\mathcal{F}}_{\mathsf{com}}, \hat{\gamma})$*-PKE assumptions hold, then the shuffle argument is adaptively computationally culpably sound w.r.t. the language* $\mathcal{R}_{sh,n}^{\mathsf{guilt}}$ *and an argument of knowledge.*

2.3.5 Efficiency

When using a Barreto-Naehrig curve [2], exponentiations in $\mathbb{G}_1$ are three times cheaper than in $\mathbb{G}_2$. Moreover, a single $(N+1)$-wide multi-exponentiations is considerably cheaper than $N+1$ exponentiations. Hence, we compute separately the number of exponentiations and multi-exponentiations in both $\mathbb{G}_1$ and $\mathbb{G}_2$ [37, 34]. For the sake of the simplicity, Prop. 3 only summarizes those numbers.

Proposition 3. *The prover's CRS consists of* $6n+7$ *elements of* $\mathbb{G}_1$ *and* $2n+4$ *elements of* $\mathbb{G}_2$*. The verifier's CRS consists of* 4 *elements of* $\mathbb{G}_1$*,* $2n+8$ *elements of* $\mathbb{G}_2$*, and* 1 *element of* $\mathbb{G}_T$*. The total CRS is* $6n+8$ *elements of* $\mathbb{G}_1$*,* $2n+8$ *elements of* $\mathbb{G}_2$*, and* 1 *element of* $\mathbb{G}_T$*, in total* $8n+17$ *group elements. The communication complexity is* $5n+2$ *elements of* $\mathbb{G}_1$ *and* $2n$ *elements of* $\mathbb{G}_2$*, in total* $7n+2$ *group elements. The prover's and the verifier's computational complexity are as in Table 2.1.*

Importantly, both the proving and verification algorithm of the new shuffle argument can be divided into offline (independent of the common input $(\mathsf{pk}, (\mathfrak{v}_i, \mathfrak{v}'_i)_{i=1}^n)$) and online (dependent on the common input) parts. The prover can precompute all elements of π_{sh} except $\mathfrak{U}$ (i.e., execute all steps of the proving algorithm, except step (9)), and send them to the verifier before the inputs are fixed. The verifier can verify $\pi_{sh} \setminus \{\mathfrak{U}\}$ (i.e., execute all steps of the verification algorithm, except step (6)) in the precomputation step. Thus, the online computational complexity is dominated by two $(n+1)$-wide multi-exponentiations for the prover, and $8n+4$ pairings for the verifier (note that $\hat{e}((\mathfrak{g}_1, \mathfrak{h}), \mathfrak{d}_2^{\gamma})$ and $\hat{e}((\mathfrak{g}_1, \mathfrak{h}), \hat{\mathfrak{d}}_2^{\hat{\gamma}})$ can also be precomputed by the verifier).

Low online complexity is highly important in e-voting, where the online time (i.e., the time interval after the ballots are gathered and before the election results are announced) can be limited for legal reasons. In this case, the mix servers can execute all but step (9) of the proving algorithm and step (6) of the verification algorithm before the votes are even cast, assuming one is able to set a priori a reasonable upper bound on n, the number of votes. See [38] for additional motivation.

2.4 FLZ shuffle argument

In the current section, we will give a full description of the new shuffle argument given in [18], followed by its efficiency analysis. Intuition behind its soundness will be given in Sect. 2.4.2. The full soundness proof is long, and omitted here, the same for the proof of zero knowledge property.

Let $\Pi = (\mathsf{genpkc}, \mathsf{enc}, \mathsf{dec})$ be an additively homomorphic cryptosystem with randomizer space R; we assume henceworth that one uses the validity-enhanced ILin cryptosystem. Consider group $\mathbb{G}_k$, $k \in \{1,2\}$. In this cryptosystem, where the secret key is $\mathsf{sk} = \gamma \leftarrow_r \mathbb{Z}_q \setminus \{0,-1\}$, the public key is $\mathsf{pk}_k \leftarrow (\mathfrak{g}_k, \mathfrak{h}_k) = (\mathfrak{g}_k, \mathfrak{g}_k^\gamma)$, and the encryption of a small $m \in \mathbb{Z}_q$ is

$$\mathsf{enc}_{\mathsf{pk}_k}(m; \vec{s}) := (\mathfrak{h}_k^{s_1}, (\mathfrak{g}_k\mathfrak{h}_k)^{s_2}, \mathfrak{g}_k^m \mathfrak{g}_k^{s_1+s_2})$$

for $\vec{s} \leftarrow_r \mathbb{Z}_q^{1\times 2}$. Denote $\mathfrak{P}_{k1} := (\mathfrak{h}_k, \mathfrak{1}_k, \mathfrak{g}_k)$ and $\mathfrak{P}_{k2} := (\mathfrak{1}_k, \mathfrak{g}_k\mathfrak{h}_k, \mathfrak{g}_k)$, thus $\mathsf{enc}_{\mathsf{pk}_k}(m; \vec{s}) = (\mathfrak{1}_k, \mathfrak{1}_k, \mathfrak{g}_k^m) \cdot \mathfrak{P}_{k1}^{s_1}\mathfrak{P}_{k2}^{s_2}$. Given $\vec{\mathfrak{v}} \in \mathbb{G}_k^3$, the decryption sets

$$\mathsf{dec}_{\mathsf{sk}}(\vec{\mathfrak{v}}) := \log_{\mathfrak{g}_k}(\mathfrak{v}_3\mathfrak{v}_2^{-1/(\gamma+1)}\mathfrak{v}_1^{-1/\gamma}) \ ,$$

Decryption succeeds since $\mathfrak{v}_3\mathfrak{v}_2^{-1/(\gamma+1)}\mathfrak{v}_1^{-1/\gamma} = \mathfrak{g}_k^m\mathfrak{g}_k^{s_1+s_2} \cdot (\mathfrak{g}_k\mathfrak{h}_k)^{-s_2/(\gamma+1)} \cdot \mathfrak{h}_k^{-s_1/\gamma} = \mathfrak{g}_k^m\mathfrak{g}_k^{s_1+s_2} \cdot \mathfrak{g}_k^{-s_2/(\gamma+1)}\mathfrak{g}_k^{-s_2\cdot\gamma/(\gamma+1)} \cdot \mathfrak{g}_k^{-s_1} = \mathfrak{g}_k^m$. This cryptosystem is CPA-secure under the 2-Incremental Linear (2-ILin) assumption, see [16]. The ILin cryptosystem is *blindable*, $\mathsf{enc}_{\mathsf{pk}_k}(m; \vec{s}) \cdot \mathsf{enc}_{\mathsf{pk}_k}(0; \vec{s}') = \mathsf{enc}_{\mathsf{pk}}(m; \vec{s} + \vec{s}')$.

The authors use a variant of the ILin cryptosystem where each plaintext is encrypted twice, in group $\mathbb{G}_1$ and in $\mathbb{G}_2$ (but by using the same secret key an the same randomizer $\vec{s}$ in both). For technical reasons (relevant to the shuffle argument but not to the ILin cryptosystem), in group $\mathbb{G}_1$ [18] uses an auxiliary generator $\hat{\mathfrak{g}}_1 = \mathfrak{g}_1^{\varrho/\beta}$ instead of $\mathfrak{g}_1$, for $(\varrho, \beta) \leftarrow_r (\mathbb{Z}_q \setminus \{0\})^2$; both encryption and decryption are done as before but just using the secret key $\mathsf{sk} = (\varrho, \beta, \gamma)$ and the public key $\mathsf{pk}_1 = (\hat{\mathfrak{g}}_1, \mathfrak{h}_1 = \hat{\mathfrak{g}}_1^\gamma)$; this also redefines $\mathfrak{P}_{k1}$. That is, $\mathsf{enc}_{\mathsf{pk}}(m; \vec{s}) = (\mathsf{enc}_{\mathsf{pk}_1}(m; \vec{s}), \mathsf{enc}_{\mathsf{pk}_2}(m; \vec{s}))$, where $\mathsf{pk}_1 = (\hat{\mathfrak{g}}_1, \mathfrak{h}_1 = \hat{\mathfrak{g}}_1^\gamma)$, and $\mathsf{pk}_2 = (\mathfrak{g}_2, \mathfrak{h}_2 = \mathfrak{g}_2^\gamma)$, and $\mathsf{dec}_{\mathsf{sk}}(\vec{\mathfrak{v}}) := \log_{\hat{\mathfrak{g}}_1}(\mathfrak{v}_3\mathfrak{v}_2^{-1/(\gamma+1)}\mathfrak{v}_1^{-1/\gamma}) = \log_{\mathfrak{g}_1}(\mathfrak{v}_3\mathfrak{v}_2^{-1/(\gamma+1)}\mathfrak{v}_1^{-1/\gamma})/(\varrho/\beta)$ for $\vec{\mathfrak{v}} \in \mathbb{G}_1^3$. They call this the *validity-enhanced* ILin cryptosystem.

Assume that $\vec{\mathfrak{v}}_i$ and $\vec{\mathfrak{v}}'_i$ are valid ciphertexts of Π. In a shuffle argument, the prover aims to convince the verifier in zero-knowledge that given $(\mathsf{pk}, (\vec{\mathfrak{v}}_i, \vec{\mathfrak{v}}'_i)_{i=1}^n)$, he knows a permutation $\sigma \in S_n$ and randomizers s_{ij}, $i \in [1\,..\,n]$ and $j \in [1\,..\,2]$, such that $\vec{\mathfrak{v}}'_i = \vec{\mathfrak{v}}_{\sigma(i)} \cdot \mathsf{enc}_{\mathsf{pk}}(0; \vec{s}_i)$ for $i \in [1\,..\,n]$. This defines the group-specific binary relation $\mathcal{R}_{sh,n}$ exactly as in [26, 30]:

$$\mathcal{R}_{sh,n} := \Big(\big(\mathsf{gk}, (\mathsf{pk}, \vec{\mathfrak{v}}_i, \vec{\mathfrak{v}}'_i)_{i=1}^n\big), (\sigma, \vec{s})\Big) : \sigma \in S_n \wedge \vec{s} \in R^{n\times 2} \wedge \big(\forall i : \vec{\mathfrak{v}}'_i = \vec{\mathfrak{v}}_{\sigma(i)} \cdot \mathsf{enc}_{\mathsf{pk}}(0; \vec{s}_i)\big) \quad \Big) \ .$$

See Prot. 1 for the full description of the shuffle argument.

Fauzi et al. (2016) note that in the real mix network, (γ, ϱ, β) is handled differently (in particular, γ — and possibly ϱ/β — will be known to the decrypting party while (ϱ, β) does not have to be known to anybody) than the real trapdoor (χ, α) that enables one to simulate the argument and thus cannot be known to anybody. Moreover, $(\mathfrak{g}_1, \mathfrak{g}_2)^{\sum P_i(\chi)}$ is in the CRS only to optimize computation. A precise efficiency analysis of this argument is given in full version of the [18].

In the rest of this section, we will explain the notion of batching and define non-batched versions (that are easier to read and analyse in the soundness proof) of the verification equations. We then state the main security theorem.

2.4.1 Batching

The authors assume that verifier checks that the batched version [4] of the equations (given in Prot. 1) hold. However, for soundness they need that the individual (non-batched) verification

$\mathsf{gencrs}(1^\kappa, n \in \mathrm{poly}(\kappa))$**:** Call $\mathsf{gk} = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}) \leftarrow \mathsf{genbp}(1^\kappa)$. Let $P_i(X)$ for $i \in [0\,..\,n]$ be polynomials (for a proper choice of polynomials see full version of the paper). Set $\vec{\chi} = (\chi, \alpha, \varrho, \beta, \gamma) \leftarrow_r \mathbb{Z}_q^2 \times (\mathbb{Z}_q \setminus \{0\})^2 \times (\mathbb{Z}_q \setminus \{0, -1\})$. Let enc be the ILin cryptosystem with the secret key γ, and let $(\mathsf{pk}_1, \mathsf{pk}_2)$ be its public key. Set

$$\mathsf{crs} \leftarrow \left(\begin{array}{l} \mathsf{gk}, (\mathfrak{g}_1^{P_i(\chi)})_{i=1}^n, \mathfrak{g}_1^\varrho, \mathfrak{g}_1^{\alpha+P_0(\chi)}, \mathfrak{g}_1^{P_0(\chi)}, (\mathfrak{g}_1^{((P_i(\chi)+P_0(\chi))^2-1)/\varrho})_{i=1}^n, \mathsf{pk}_1 = (\hat{\mathfrak{g}}_1 = \mathfrak{g}_1^{\varrho/\beta}, \mathfrak{h}_1 = \hat{\mathfrak{g}}_1^\gamma), \\ (\mathfrak{g}_2^{P_i(\chi)})_{i=1}^n, \mathfrak{g}_2^\varrho, \mathfrak{g}_2^{-\alpha+P_0(\chi)}, \mathsf{pk}_2 = (\mathfrak{g}_2, \mathfrak{h}_2 = \mathfrak{g}_2^\gamma), \mathfrak{g}_2^\beta, \hat{e}(\mathfrak{g}_1, \mathfrak{g}_2)^{1-\alpha^2}, (\mathfrak{g}_1, \mathfrak{g}_2)^{\sum_{i=1}^n P_i(\chi)} \end{array} \right).$$

and $\mathsf{td} \leftarrow (\chi, \varrho)$. Return $(\mathsf{crs}, \mathsf{td})$.

$\mathsf{pro}(\mathsf{crs}; \vec{\mathfrak{v}} \in (\mathbb{G}_1 \times \mathbb{G}_2)^{3n}; \sigma \in S_n, \vec{s} \in \mathbb{Z}_q^{n \times 2})$**:**

1. For $i = 1$ to $n-1$:
   (a) Set $r_i \leftarrow_r \mathbb{Z}_q$. Set $(\mathfrak{A}_{i1}, \mathfrak{A}_{i2}) \leftarrow (\mathfrak{g}_1, \mathfrak{g}_2)^{P_{\sigma^{-1}(i)}(\chi)+r_i\varrho}$.
2. Set $r_n \leftarrow -\sum_{i=1}^{n-1} r_i$.
3. Set $(\mathfrak{A}_{n1}, \mathfrak{A}_{n2}) \leftarrow (\mathfrak{g}_1, \mathfrak{g}_2)^{\sum_{i=1}^n P_i(\chi)} / \prod_{i=1}^{n-1} (\mathfrak{A}_{i1}, \mathfrak{A}_{i2})$.
4. For $i = 1$ to n: `/* Sparsity, for permutation matrix: */`
   (a) Set $\pi_{\mathsf{1sp}:i} \leftarrow (\mathfrak{A}_{i1}\mathfrak{g}_1^{P_0(\chi)})^{2r_i} (\mathfrak{g}_1^\varrho)^{-r_i^2}\, \mathfrak{g}_1^{((P_{\sigma^{-1}(i)}(\chi)+P_0(\chi))^2-1)/\varrho}$.
5. For $i = 1$ to n: `/* Shuffling itself */`
   (a) Set $(\vec{\mathfrak{v}}'_{i1}, \vec{\mathfrak{v}}'_{i2}) \leftarrow (\vec{\mathfrak{v}}_{\sigma(i)1}, \vec{\mathfrak{v}}_{\sigma(i)2}) \cdot (\mathsf{enc}_{\mathsf{pk}_1}(0; \vec{s}_i), \mathsf{enc}_{\mathsf{pk}_2}(0; \vec{s}_i))$.
6. Set `/* Consistency */`
   (a) For $k = 1$ to 2: Set $r_{s:k} \leftarrow_r \mathbb{Z}_q$. Set $\pi_{\mathsf{c1}:k} \leftarrow \mathfrak{g}_2^{\sum_{i=1}^n s_{ik} P_i(\chi) + r_{s:k}\varrho}$.
   (b) $(\vec{\pi}_{\mathsf{c2}:1}, \vec{\pi}_{\mathsf{c2}:2}) \leftarrow \prod_{i=1}^n (\vec{\mathfrak{v}}_{i1}, \vec{\mathfrak{v}}_{i2})^{r_i} \cdot (\mathsf{enc}_{\mathsf{pk}_1}(0; \vec{r}_s), \mathsf{enc}_{\mathsf{pk}_2}(0; \vec{r}_s))$.
7. Return $\pi_{sh} \leftarrow (\vec{\mathfrak{v}}', (\mathfrak{A}_{i1}, \mathfrak{A}_{i2})_{i=1}^{n-1}, (\pi_{\mathsf{1sp}:i})_{i=1}^n, \pi_{\mathsf{c1}:1}, \pi_{\mathsf{c1}:2}, \vec{\pi}_{\mathsf{c2}:1}, \vec{\pi}_{\mathsf{c2}:2})$.

$\mathsf{ver}(\mathsf{crs}; \vec{\mathfrak{v}}; \vec{\mathfrak{v}}', (\mathfrak{A}_{i1}, \mathfrak{A}_{i2})_{i=1}^{n-1}, (\pi_{\mathsf{1sp}:i})_{i=1}^n, \pi_{\mathsf{c1}:1}, \pi_{\mathsf{c1}:2}, \vec{\pi}_{\mathsf{c2}:1}, \vec{\pi}_{\mathsf{c2}:2})$**:**

1. Set $(\mathfrak{A}_{n1}, \mathfrak{A}_{n2}) \leftarrow (\mathfrak{g}_1, \mathfrak{g}_2)^{\sum_{i=1}^n P_i(\chi)} / \prod_{i=1}^{n-1} (\mathfrak{A}_{i1}, \mathfrak{A}_{i2})$.
2. Set $(p_{1i}, p_{2j}, p_{3ij}, p_{4j})_{i \in [1\,..\,n], j \in [1\,..\,3]} \leftarrow_r \mathbb{Z}_q^{4n+6}$.
3. Check that `/* Permutation matrix: */`
   $\prod_{i=1}^n \hat{e}\left((\mathfrak{A}_{i1}\mathfrak{g}_1^{\alpha+P_0(\chi)})^{p_{1i}}, \mathfrak{A}_{i2}\mathfrak{g}_2^{-\alpha+P_0(\chi)} \right) = \hat{e}\left(\prod_{i=1}^n \pi_{\mathsf{1sp}:i}^{p_{1i}}, \mathfrak{g}_2^\varrho \right) \cdot \hat{e}(\mathfrak{g}_1, \mathfrak{g}_2)^{(1-\alpha^2)\sum_{i=1}^n p_{1i}}$.
4. Check that `/* Validity: */`
   $\hat{e}\left(\mathfrak{g}_1^\varrho, \prod_{j=1}^3 \pi_{\mathsf{c2}:2j}^{p_{2j}} \cdot \prod_{i=1}^n \prod_{j=1}^3 (\mathfrak{v}'_{i2j})^{p_{3ij}} \right) = \hat{e}\left(\prod_{j=1}^3 \pi_{\mathsf{c2}:1j}^{p_{2j}} \cdot \prod_{i=1}^n \prod_{j=1}^3 (\mathfrak{v}'_{i1j})^{p_{3ij}}, \mathfrak{g}_2^\beta \right)$.
5. Set $\mathfrak{R} \leftarrow \hat{e}\left(\hat{\mathfrak{g}}_1, \pi_{\mathsf{c1}:2}^{p_{42}} (\pi_{\mathsf{c1}:1}\pi_{\mathsf{c1}:2})^{p_{43}}\right) \cdot \hat{e}\left(\mathfrak{h}_1, \pi_{\mathsf{c1}:1}^{p_{41}} \pi_{\mathsf{c1}:2}^{p_{42}}\right) / \hat{e}\left(\prod_{j=1}^3 \pi_{\mathsf{c2}:1j}^{p_{4j}}, \mathfrak{g}_2^\varrho \right)$.
6. Check that `/* Consistency: */`
   $\prod_{i=1}^n \hat{e}\left(\prod_{j=1}^3 (\mathfrak{v}'_{i1j})^{p_{4j}}, \mathfrak{g}_2^{P_i(\chi)} \right) / \prod_{i=1}^n \hat{e}\left(\prod_{j=1}^3 \mathfrak{v}_{i1j}^{p_{4j}}, \mathfrak{A}_{i2} \right) = \mathfrak{R}$.

Protocol 1: The new shuffle argument

equations hold. They will show that we still have soundness even if the verifier checks batched versions of the equations.

Lemma 4. *Assume $(p_i)_{i\in[1\,..\,k]}$ are values chosen uniformly random from $\mathbb{Z}_q^k$. Assume $\vec{\chi}$ are values chosen uniformly at random from $\mathbb{Z}_q$. Assume f_i are some polynomials of degree* $\mathrm{poly}(\kappa)$*. If the equation $\prod_{i=1}^{k} \hat{e}(\mathfrak{g}_1, \mathfrak{g}_2)^{f_i(\vec{\chi})p_i} = \mathfrak{1}_T$ holds, then with probability $\geq 1 - 1/q$ the k pairing equations $\hat{e}(\mathfrak{g}_1, \mathfrak{g}_2)^{f_i(\vec{\chi})} = \mathfrak{1}_T$, $i \in [1\,..\,k]$ also hold.*

The following corollary follows immediately from Lem. 4.

Corollary 5. *Assume $\vec{\chi} = (\chi, \alpha, \varrho, \beta, \gamma)$ is chosen uniformly random from $\mathbb{Z}_q^2 \times (\mathbb{Z}_q \setminus \{0\})^2 \times (\mathbb{Z}_q \setminus \{0, -1\})$. Assume $(p_{1i}, p_{2j}, p_{3ij}, p_{4j})_{i\in[1\,..\,n],j\in[1\,..\,3]}$ are values chosen uniformly random from $\mathbb{Z}_q^{4n+6}$. Consider the verification steps in Prot. 1.*

- *If the verification on Step 3 accepts, then (with probability $\geq 1 - 1/q$) for $i \in [1\,..\,n]$,*

$$\hat{e}\left(\mathfrak{A}_{i1}\mathfrak{g}_1^{\alpha+P_0(\chi)}, \mathfrak{A}_{i2}\mathfrak{g}_2^{-\alpha+P_0(\chi)}\right) = \hat{e}\left(\pi_{\mathsf{1sp}:i}, \mathfrak{g}_2^{\varrho}\right)\hat{e}(\mathfrak{g}_1, \mathfrak{g}_2)^{1-\alpha^2} \quad . \tag{2.2}$$

- *If the verification on Step 4 accepts, then with probability $\geq 1 - 1/q$,*

$$\hat{e}(\mathfrak{g}_1^{\varrho}, \pi_{\mathsf{c2}:2i}) = \hat{e}(\pi_{\mathsf{c2}:1i}, \mathfrak{g}_2^{\beta}) \quad , \quad i \in [1\,..\,3] \quad , \tag{2.3}$$

$$\hat{e}\left(\mathfrak{g}_1^{\varrho}, \mathfrak{v}'_{i2j}\right) = \hat{e}(\mathfrak{v}'_{i1j}, \mathfrak{g}_2^{\beta}) \quad , \quad i \in [1\,..\,n], j \in [1\,..\,3] \quad . \tag{2.4}$$

- *If the verification on Step 6 accepts, then with probability $\geq 1 - 1/q$,*

$$\prod_{i=1}^{n} \hat{e}\left(\vec{\mathfrak{v}}'_{i1}, \mathfrak{g}_2^{P_i(\chi)}\right) / \prod_{i=1}^{n} \hat{e}\left(\vec{\mathfrak{v}}_{i1}, \mathfrak{A}_{i2}\right) = \hat{e}\left(\mathfrak{P}_{11}, \pi_{\mathsf{c1}:1}\right)\hat{e}\left(\mathfrak{P}_{12}, \pi_{\mathsf{c1}:2}\right)/\hat{e}\left(\vec{\pi}_{\mathsf{c2}:1}, \mathfrak{g}_2^{\varrho}\right) \quad . \tag{2.5}$$

This means that with probability $\geq 1-3/q$, checking the batched version of verification equations (as in Prot. 1) is equivalent to the checking of individual verification equations (as in Cor. 5).

The authors note that Cor. 5 also holds when $\vec{\chi}$ is chosen according to the distribution, stipulated in Prot. 1.

2.4.2 Intuition behind soundness

Throughout this paper, we follow notion from [18] and use a variation of the polynomial commitment scheme of type $\mathsf{com}_j(P_i; \vec{a}; r) := \mathfrak{h}^{\sum_{i=1}^{n} a_i P_i(\chi) + r\varrho}$, where $\mathfrak{h}$ is a generator of $\mathbb{G}_j$, χ and ϱ are random values from $\mathbb{Z}_q$, and $P_i(X)$ are well-chosen polynomials. (The choice of $P_i(X)$ is fixed by the 1-sparsity argument) Several variants of this commitment scheme are well-known to be perfectly hiding and computationally binding (under a suitable computational assumption, security of which is usually proved in the GBGM, [25, 28]). However, since [18] only relies on the security of this commitment scheme within the GBGM soundness proof of the shuffle, they state neither the concrete assumption nor the security requirements (like hiding and binding) of a commitment scheme.

On the last three steps, see Prot. 1, the verifier executes four different verifications, restated in an easier to read format in Cor. 5. Each of these verifications has an intuitive meaning, resulting in a different subargument. However, since all of them have to use the same CRS and the soundness proof is in the GBGM, the subarguments interact strongly.

Soundness proof in the GBGM uses the following idea. An adversary can only produce group elements from $\mathbb{G}_1$ or $\mathbb{G}_2$ that are products of the elements of the same group given in the CRS; elements of $\mathbb{G}_T$ can also be output by the pairing operation. Let $\vec{\chi} = (\chi, \alpha, \varrho, \beta, \gamma)$ be concrete (randomly chosen) values from $\mathbb{Z}_q$ and $\vec{X} = (X, X_\alpha, X_\varrho, X_\beta, X_\gamma)$ be the corresponding random

variables. E.g., if $\mathcal{F}(\vec{X}) = \{F_i(\vec{X})\}$ is the set of all rational functions such that $\mathfrak{g}_1^{\mathcal{F}(\vec{\chi})} = \{\mathfrak{g}_1^{F_i(\vec{\chi})}\}$ is equal to the set of all CRS values in $\mathbb{G}_1$, then any value that the adversary creates in $\mathbb{G}_1$ must be of the form $\mathfrak{g}_1^{A(\vec{\chi})}$, where $A(\vec{X}) \in \operatorname{span} \mathcal{F}(\vec{X})$.

In this way, after taking a discrete logarithm, each verification equation can be written in the form $\mathcal{V}(\vec{\chi}) = 0$ for some polynomial $\mathcal{V}(\vec{X})$ known to the adversary. However, since the values in $\vec{\chi}$ were chosen uniformly random, from the Schwartz-Zippel lemma [35, 39] the authors conclude that $\mathcal{V}(\vec{X}) = 0$ as a polynomial (or a rational function), except with negligible probability $O(n)/q$. From $\mathcal{V}(\vec{X}) = 0$, they deduce that all the coefficients of terms $X_\alpha^{i_1} X_\varrho^{i_2} X_\beta^{i_3} X_\gamma^{i_4}$ in $\mathcal{V}(\vec{X}) \cdot \mathcal{V}^*(\vec{X})$ (where $\mathcal{V}^*(\vec{X})$ is the denominator of $\mathcal{V}(\vec{X})$) are zero, giving us several equations related to the adversary's chosen values. From these equations and the linear independence of polynomials $P_i(X)$, they deduce that the adversary's chosen values must be of a certain form, except with negligible probability $O(n)/q$.

More precisely, for symbolic values T and t, define (by following the definition of the CRS in Prot. 1)

$$\begin{aligned}
\mathsf{crs}_1(\vec{X}, T, t) =& t(X) + T_\varrho X_\varrho + T_\alpha \cdot (X_\alpha + P_0(X)) + T_0 P_0(X) + \frac{t^\dagger(X) Z(X)}{X_\varrho} + \frac{T_{\varrho\beta} X_\varrho}{X_\beta} + \frac{T_\gamma X_\varrho X_\gamma}{X_\beta} \ , \\
\mathsf{crs}_2(\vec{X}, T, t) =& t(X) + T_\varrho X_\varrho + T_\alpha \cdot (-X_\alpha + P_0(X)) + T_1 + T_\gamma X_\gamma + T_\beta X_\beta \ ,
\end{aligned}$$

where $t^\dagger(X)$ is in the span of $\{((P_i(X) + P_0(X)) - 1)^2 / Z(X)\}_{i=1}^n$ and $t(X)$ is in the span of $\{P_i(X)\}_{i=1}^n$. In particular, all "daggered" polynomials (e.g., $b^\dagger(X)$) are in the span of $\{((P_i(X) + P_0(X)) - 1)^2 / Z(X)\}_{i=1}^n$. Since $\deg Z(X) = n + 1$, $\deg t^\dagger(X) \leq n - 1$, and $\deg t(X) \leq n$, then $\deg(\mathsf{crs}_1(\vec{X}, T, t) \cdot X_\varrho X_\beta) \leq (n-1) + (n+1) - 1 + 2 = 2n + 1$. (Multiplication with $X_\varrho X_\beta$ is needed to make $\mathsf{crs}_1(\vec{X}, T, t)$ a polynomial.) Analogously, $\deg \mathsf{crs}_2(\vec{X}, T, t) \leq n$. Importantly, $\{P_i(X)\}_{i=0}^n$ is linearly independent. In particular, $P_0(X)$ is linearly independent to all other polynomials present in $\mathsf{crs}_1(\vec{X})$ and $\mathsf{crs}_2(\vec{X})$, except the "daggered" polynomial $t^\dagger(X)$.

Since the shuffle argument adversary is a GBGM adversary (and one uses lLin encryption), she knows the following polynomials (in the case of crs_2-functions), Laurent polynomials (in the case of crs_1-functions) or rational functions (in the case of $M_{ij}(\vec{X})$, $M'_{ij}(\vec{X})$, and $M_{E:j}(\vec{X})$), where $\hat{\mathfrak{g}}_2 = \mathfrak{g}_2$:

$$\begin{aligned}
A(\vec{X}) =& \mathsf{crs}_1(\vec{X}, A, a) && \text{s.t. } \mathfrak{A}_1 = \mathfrak{g}_1^{A(\vec{\chi})} \ , \\
B(\vec{X}) =& \mathsf{crs}_2(\vec{X}, B, b) && \text{s.t. } \mathfrak{A}_2 = \mathfrak{g}_2^{B(\vec{\chi})} \ , \\
C(\vec{X}) =& \mathsf{crs}_1(\vec{X}, C, c) && \text{s.t. } \pi_{\mathsf{1sp}} = \mathfrak{g}_1^{C(\vec{\chi})} \ , \\
D_j(\vec{X}) =& \mathsf{crs}_2(\vec{X}, D_j, d_j) && \text{s.t. } \pi_{\mathsf{c1}:j} = \mathfrak{g}_2^{D_j(\vec{\chi})} \ , \\
E_{kj}(\vec{X}) =& \mathsf{crs}_k(\vec{X}, E_{kj}, e_{kj}) && \text{s.t. } \pi_{\mathsf{c2}:kj} = \mathfrak{g}_k^{E_{kj}(\vec{\chi})} \ , \\
V_{ikj}(\vec{X}) =& \mathsf{crs}_k(\vec{X}, V_{ikj}, v_{ikj}) && \text{s.t. } \mathfrak{v}_{ikj} = \hat{\mathfrak{g}}_k^{V_{ikj}(\vec{\chi})} \ , \\
V'_{ikj}(\vec{X}) =& \mathsf{crs}_k(\vec{X}, V'_{ikj}, v'_{ikj}) && \text{s.t. } \mathfrak{v}'_{ikj} = \hat{\mathfrak{g}}_k^{V'_{ikj}(\vec{\chi})} \ , \\
M_{ij}(\vec{X}) =& V_{ij3}(\vec{X}) - V_{ij2}(\vec{X})/(X_\gamma + 1) - V_{ij1}/X_\gamma && \text{s.t. } \mathsf{dec}_{\mathsf{sk}}(\vec{\mathfrak{v}}_{ij}) = M_{ij}(\vec{\chi}) \ , \\
M'_{ij}(\vec{X}) =& V'_{ij3}(\vec{X}) - V'_{ij2}(\vec{X})/(X_\gamma + 1) - V'_{ij1}/X_\gamma && \text{s.t. } \mathsf{dec}_{\mathsf{sk}}(\vec{\mathfrak{v}}'_{ij}) = M'_{ij}(\vec{\chi}) \ , \\
M_{E:j}(\vec{X}) =& E_{j3}(\vec{X}) - E_{j2}(\vec{X})/(X_\gamma + 1) - E_{j1}/X_\gamma && \text{s.t. } \mathsf{dec}_{\mathsf{sk}}(\vec{\pi}_{\mathsf{c2}:j}) = M_{E:j}(\vec{\chi}) \ . \qquad (2.6)
\end{aligned}$$

The authors note that a major obstacle in proving soundness in the GBGM is that all subarguments must use the same CRS. In particular, a subargument that is sound by itself might stop

being sound due to the elements in the CRS that are added because of other subarguments. They tackle this problem by introducing random variables α (that is only needed in Eq. (2.2)) and β (that is needed in Eq. (2.3) and Eq. (2.4)).

Briefly, the verifier makes three checks. Eq. (2.2), the "permutation matrix argument", guarantees that the prover has committed to a permutation matrix corresponding to some permutation σ. Eq. (2.4) and Eq. (2.3), the "validity argument", guarantee that the ciphertexts have not been formed in a devious way that would make the consistency argument to be unsound. Eq. (2.5), the "consistency argument", guarantees that the prover has used the same permutation σ to shuffle the ciphertexts.

2.4.3 Permutation matrix argument

Consider the subargument of Prot. 1, where the verifier just computes $(\mathfrak{A}_{n1}, \mathfrak{A}_{n2})$ and then performs the verification Eq. (2.2) for each $i = 1$ to n. The authors call it the *permutation matrix argument*. This name is motivated by showing that after the permutation matrix argument only, the verifier is convinced that $(\mathfrak{A}_{11}, \ldots, \mathfrak{A}_{n1})$ commits to a permutation matrix. For this, the authors first prove the security of its subargument — the 1-sparsity argument [30] — where the verifier performs the verification Eq. (2.2) for exactly one i.

To prove the security of permutation matrix argument, [18] solves a quite complicated system of polynomial equations. They do it by using a computer algebra system, see full version for more details.

2.4.4 Validity argument

As a subroutine in the argument, [18] makes the verifier check the validity of all ciphertexts. This is done by checking Eq. (2.4) (and Eq. (2.3)). The main goal of the validity check is to show that the prover did not use "forbidden" terms $\mathfrak{g}_k^{P_i(\chi)}$ and $\mathfrak{g}_i^{\varrho}$ when computing the ciphertexts $\mathfrak{v}'_{ik}$ and $\vec{\pi}_{\mathsf{c2}:k}$. In the case of the ILin cryptosystem, the validity argument provides a proof that both $\vec{\mathfrak{v}}'_{i1}$ and $\vec{\mathfrak{v}}'_{i2}$ decrypt to a plaintext of form $M_i(\vec{X}) = \sum M_{ij} f_{ij}(\vec{X})$, for known coefficients M_{ij} and polynomials $f_{ij}(\vec{X})$, where none of the rational functions f_{ij} depends on either X or X_ϱ. Similar assurance is provided about the plaintext hidden in $\vec{\pi}_{\mathsf{c2}:k}$. Employing validity subarguments allows the consistency subargument to be more efficient than in [26, 17].

2.4.5 Consistency argument

Finally, [18] shows that performing all checks guarantees that $\mathsf{dec}_{\mathsf{sk}}(\vec{\mathfrak{v}}'_i) = \mathsf{dec}_{\mathsf{sk}}(\vec{\mathfrak{v}}_{\sigma(i)}) \neq \perp$ for some permutation $\sigma \in S_n$. The main observation is that a permutation of ciphertexts (without rerandomization) is invariant under multiplication: without rerandomizing the ciphertexts, the (non-batched) verification Eq. (2.5) would just be the identity $\hat{e}(\vec{\mathfrak{v}}'_{i1}, \mathfrak{g}_2^{P_i(\chi)}) = \hat{e}(\vec{\mathfrak{v}}_{i1}, \mathfrak{g}_2^{P_{\sigma^{-1}(i)}(\chi)})$, for all i. However, this trivially leaks the permutation σ, and hence is not secure. To ensure privacy, $\vec{\mathfrak{v}}'_{i1}$ must be rerandomized, and $\mathfrak{g}_2^{P_{\sigma^{-1}(i)}(\chi)}$ must be replaced by a commitment to the unit vector $\vec{e}_{\sigma^{-1}(i)}$. This makes the final verification slightly more complicated.

A version of Eq. (2.5) was also used in [26, 30, 17]. However, the shuffle arguments from [26, 17] need to execute two versions of Eq. (2.5), once with $P_i(X)$ and once with different carefully chosen polynomials $\hat{P}_i(X)$ in $\mathbb{G}_2$. (See [26, 17] for an explanation.) In addition, one must prove that those two versions are consistent between each other (by providing a same-message argument, in the terminology of [17]). This makes the arguments of [26, 17] quite complicated.

Similarly to [30], Fauzi et al. (2016) avoid this complication by having a validity argument on the ciphertexts. Since valid ciphertexts are not dependent of $P_i(X)$, it suffices for the verifier to execute just one version of Eq. (2.5).

Bibliography

[1] M. Ambrona, G. Barthe, and B. Schmidt. Automated Unbounded Analysis of Cryptographic Constructions in the Generic Group Model. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 822–851, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg.

[2] P. S. L. M. Barreto and M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In B. Preneel and S. E. Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331, Kingston, ON, Canada, Aug. 11–12, 2005. Springer, Heidelberg.

[3] G. Barthe, E. Fagerholm, D. Fiore, A. Scedrov, B. Schmidt, and M. Tibouchi. Strongly-Optimal Structure Preserving Signatures from Type II Pairings: Synthesis and Lower Bounds. In J. Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 355–376, Gaithersburg, Maryland, USA, March 30–April 1, 2015. Springer, Heidelberg.

[4] M. Bellare, J. A. Garay, and T. Rabin. Batch Verification with Applications to Cryptography and Checking. In C. L. Lucchesi and A. V. Moura, editors, *LATIN 1998*, volume 1380 of *LNCS*, pages 170–191, Campinas, Brazil, Apr. 20–24, 1998. Springer, Heidelberg.

[5] N. Bitansky, R. Canetti, O. Paneth, and A. Rosen. On the Existence of Extractable One-Way Functions. In D. Shmoys, editor, *STOC 2014*, pages 505–514, New York, NY, USA, May 31 – Jun 3, 2014. ACM Press.

[6] N. Bitansky, D. Dachman-Soled, S. Garg, A. Jain, Y. T. Kalai, A. Lopez-Alt, and D. Wichs. Why "Fiat-Shamir for Proofs" Lacks a Proof. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 182–201, Tokyo, Japan, Mar. 3–6, 2013. Springer, Heidelberg.

[7] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero-Knowledge and Its Applications. In *STOC 1988*, pages 103–112, Chicago, Illinois, USA, May 2–4, 1988. ACM Press.

[8] D. Boneh, X. Boyen, and E.-J. Goh. Hierarchical Identity Based Encryption with Constant Size Ciphertext. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 440–456, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg.

[9] B. Buchberger. *An Algorithm for Finding the Basis Elements of the Residue Class Ring of a Zero Dimensional Polynomial Ideal.* PhD thesis, University of Innsbruck, 1965.

[10] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. In J. S. Vitter, editor, *STOC 1998*, pages 209–218, Dallas, Texas, USA, May 23–26, 1998.

[11] J. H. Cheon and T. Takagi, editors. *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security,*

Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II, volume 10032 of *Lecture Notes in Computer Science*, 2016.

[12] M. Ciampi, G. Persiano, L. Siniscalchi, and I. Visconti. A Transform for NIZK Almost as Efficient and General as the Fiat-Shamir Transform Without Programmable Random Oracles. In E. Kushilevitz and T. Malkin, editors, *TCC 2016-A (2)*, volume 9563 of *LNCS*, pages 83–111, Tel Aviv, Israel, Jan. 10–13, 2016. Springer, Heidelberg.

[13] G. Danezis, C. Fournet, J. Groth, and M. Kohlweiss. Square Span Programs with Applications to Succinct NIZK Arguments. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014 (1)*, volume 8873 of *LNCS*, pages 532–550, Kaohsiung, Taiwan, R.O.C., Dec. 7–11, 2014. Springer, Heidelberg.

[14] A. W. Dent. Adapting the Weaknesses of the Random Oracle Model to the Generic Group Model. In Y. Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 100–109, Queenstown, New Zealand, Dec. 1–5, 2002. Springer, Heidelberg.

[15] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Trans. on Inf. Theory*, 31(4):469–472, 1985.

[16] A. Escala, G. Herold, E. Kiltz, C. Ràfols, and J. L. Villar. An Algebraic Framework for Diffie-Hellman Assumptions. In R. Canetti and J. Garay, editors, *CRYPTO (2) 2013*, volume 8043 of *LNCS*, pages 129–147, Santa Barbara, California, USA, Aug. 18–22, 2013. Springer, Heidelberg.

[17] P. Fauzi and H. Lipmaa. Efficient Culpably Sound NIZK Shuffle Argument without Random Oracles. In K. Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 200–216, San Franscisco, CA, USA, February 29–March 4, 2016. Springer, Heildeberg.

[18] P. Fauzi, H. Lipmaa, and M. Zając. A Shuffle Argument Secure in the Generic Model. In Cheon and Takagi [11], pages 841–872.

[19] M. Fischlin. A Note on Security Proofs in the Generic Model. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 458–469, Kyoto, Japan, Dec. 3–7, 2001. Springer, Heidelberg.

[20] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic Span Programs and NIZKs without PCPs. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645, Athens, Greece, Apr. 26–30, 2013. Springer, Heidelberg.

[21] S. Goldwasser and Y. T. Kalai. On the (In)security of the Fiat-Shamir Paradigm. In *FOCS 2003*, pages 102–113, Cambridge, MA, USA, Oct. 11–14, 2003. IEEE, IEEE Computer Society Press.

[22] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. In R. Sedgewick, editor, *STOC 1985*, pages 291–304, Providence, Rhode Island, USA, May 6–8, 1985. ACM Press.

[23] P. Golle, S. Jarecki, and I. Mironov. Cryptographic Primitives Enforcing Communication and Storage Complexity. In M. Blaze, editor, *FC 2002*, volume 2357 of *LNCS*, pages 120–135, Southhampton Beach, Bermuda, Mar. 11–14, 2002. Springer, Heidelberg.

[24] J. Groth. A Verifiable Secret Shuffle of Homomorphic Encryptions. *J. Cryptology*, 23(4):546–579, 2010.

[25] J. Groth. Short Pairing-Based Non-interactive Zero-Knowledge Arguments. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340, Singapore, Dec. 5–9, 2010. Springer, Heidelberg.

[26] J. Groth and S. Lu. A Non-interactive Shuffle with Pairing Based Verifiability. In K. Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 51–67, Kuching, Malaysia, Dec. 2–6, 2007. Springer, Heidelberg.

[27] J. Groth, R. Ostrovsky, and A. Sahai. New Techniques for Noninteractive Zero-Knowledge. *Journal of the ACM*, 59(3), 2012.

[28] H. Lipmaa. Progression-Free Sets and Sublinear Pairing-Based Non-Interactive Zero-Knowledge Arguments. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189, Taormina, Italy, Mar. 18–21, 2012. Springer, Heidelberg.

[29] H. Lipmaa and B. Zhang. A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument. In I. Visconti and R. D. Prisco, editors, *SCN 2012*, volume 7485 of *LNCS*, pages 477–502, Amalfi, Italy, September 5–7, 2012. Springer, Heidelberg.

[30] H. Lipmaa and B. Zhang. A More Efficient Computationally Sound Non-Interactive Zero-Knowledge Shuffle Argument. *Journal of Computer Security*, 21(5):685–719, 2013.

[31] U. M. Maurer. Abstract Models of Computation in Cryptography. In N. P. Smart, editor, *Cryptography and Coding 2005*, pages 1–12, Cirencester, UK, Dec. 19–21, 2005.

[32] C. A. Neff. A Verifiable Secret Shuffle and Its Application to E-Voting. In *ACM CCS 2001*, pages 116–125, Philadelphia, Pennsylvania, USA, Nov. 6–8 2001. ACM Press.

[33] J. B. Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 111–126, Santa Barbara, USA, Aug. 18–22, 2002. Springer, Heidelberg.

[34] N. Pippenger. On the Evaluation of Powers and Monomials. *SIAM J. Comput.*, 9(2):230–250, 1980.

[35] J. T. Schwartz. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *Journal of the ACM*, 27(4):701–717, 1980.

[36] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In W. Fumy, editor, *EUROCRYPT 1997*, volume 1233 of *LNCS*, pages 256–266, Konstanz, Germany, 11–15 May 1997. Springer, Heidelberg.

[37] E. G. Straus. Addition Chains of Vectors. *Amer. Math. Monthly*, 70:806–808, 1964.

[38] D. Wikström. A Commitment-Consistent Proof of a Shuffle. In C. Boyd and J. M. G. Nieto, editors, *ACISP 2009*, volume 5594 of *LNCS*, pages 4007–421, Brisbane, Australia, July 1–3, 2009. Springer, Heidelberg.

[39] R. Zippel. Probabilistic Algorithms for Sparse Polynomials. In E. W. Ng, editor, *EUROSM 1979*, volume 72 of *LNCS*, pages 216–226, Marseille, France, June 1979. Springer, Heidelberg.

Part II

Robust and efficient decryption mix network designs

3. The Loopix anonymity system

3.1 Introduction

One of the PANORAMIX objectives is to apply the built mix network infrastracture to privacy-preserving messaging, where two or more users may communicate privately without any third party being able to track their communication, neither the content of the messages nor meta-data. In traditional communication security, the confidentiality of messages is protected through encryption, which exposes meta-data, such as who is sending messages to whom, to network eavesdroppers. As illustrated by recent leaks of extensive mass surveillance programs[1], exposing such meta-data leads to significant privacy risks.

There are only a few systems, which allow to hide both the content and the meta-data of the messages. Created in 2004 Tor [19], a circuit-based onion routing network has become the most popular anonymous communication tool. Several state-of-the-art systems, such as Herd [32], Riposte [10], HORNET [9] and Vuvuzela [40] were proposed as extension of this circuit-based paradigm. However, unless cover traffic is employed, onion routing is susceptible to traffic analysis attacks [6] by an adversary that can monitor network links between nodes (see also Chapter 5). Recent revelations suggest that capabilities of large intelligence agencies approach that of global passive observers—the most powerful form of this type of adversary. It is not sufficient to provide strong anonymity against such an adversary while providing low-latency communication. A successful system additionally needs to resist powerful active attacks and use an efficient, yet secure way of transmitting messages. Moreover, the system needs to be scalable to a large number of clients, which makes classical approaches based on synchronized rounds infeasible.

In this chapter we present Loopix, a novel anonymous communication system, based on mix networks, which is resistant against powerful adversaries who are capable of observing all communications and performing active attacks. We demonstrate that such a mix architecture can support low-latency communications that can tolerate small delays, at the cost of using some extra bandwidth for cover traffic. Delay, cover and real traffic can be flexibly traded-off against each other to offer resistance to traffic analysis. Loopix provides *'third-party'* anonymity, namely it hides the sender-receiver relationships from third parties, but senders and recipients can identify one another. This simplifies the design of the system, prevents abuse, and provides security guarantees against powerful active adversaries performing $(n-1)$ attacks [37], where the adversary blocks all but one message sent to a mix server, to trace the remaining message.

Loopix provides anonymity for private email or instant messaging applications, which are one of the main goals of the PANORAMIX project. The proposed solutions introduces a novel design of a mix network, supporting WP3 and WP4 and the WP7 use case.

[1] See EFF's guide at `https://www.eff.org/files/2014/05/29/unnecessary_and_disproportionate.pdf`

3.2 Model and goals

In this section, we first outline the design of Loopix. Then we discuss the security goals and types of adversaries against which Loopix protects its users' privacy.

3.2.1 High-level overview

Loopix is a mix network [7] based architecture allowing *users*, distinguished as *senders* and *receivers*, to route messages anonymously to each other using an infrastructure of *mix* servers, acting as relays. These mix servers are arranged in a stratified topology [20] to ensure both horizontal scalability and a sparse topology that concentrates traffic on fewer links [11]. Each user is allowed to access the Loopix network through their association with a *provider*, a special type of mix server. Each provider has a long-term relationship with its users and may authenticate them, potentially bill them or discontinue their access to the network. The provider not only serves as an access point, but also stores users' incoming messages. These messages can be retrieved at any time, hence users do not have to worry about lost messages when they are off-line. In contrast to previous anonymous messaging designs [40, 10], Loopix does not operate in deterministic rounds, but runs as a continuous system. Additionally, Loopix uses the Poisson mixing technique that is based on the independent delaying of messages, which makes the timings of packets unlinkable. This approach does not require the synchronization of client-provider rounds and does not degrade the usability of the system for temporarily off-line clients. Moreover, Loopix introduces different types of cover traffic to foil de-anonymization attacks.

3.2.2 Threat model

Loopix assumes sophisticated, strategic, and well-resourced adversaries concerned with linking users to their communications and/or their communication partner(s). As such, Loopix considers adversaries with three distinct *capabilities*, that are described next.

Firstly, a *global passive adversary* (GPA) is able to observe all network traffic between users and providers and between mix servers. This adversary is able to observe the entire network infrastructure, launch network attacks such as BGP re-routing [3] or conduct indirect observations such as load monitoring and off-path attacks [24]. Thus, the GPA is an abstraction that represents many different classes of adversaries able to observe some or all information between network nodes.

Secondly, the adversary has the ability to observe all of the internal state of some corrupted or malicious mix relays. The adversary may inject, drop, or delay messages. She also has access to, and operates, using the secrets of those compromised parties. Furthermore, such corrupted nodes may deviate from the protocol, or inject malformed messages. A variation of this ability is where the mix relay is also the provider node meaning that the adversary additionally knows the mapping between clients and their mailboxes. We say that the provider is *corrupt*, but is restricted to being honest but curious. In Loopix, we assume that a fraction of mix/provider relays can be corrupted or are operated by the adversary.

Finally, the adversary has the ability to participate in the Loopix system as a compromised user, who may deviate from the protocol. We assume that the adversary can control a limited number of such users—excluding Sybil attacks [21] from the Loopix threat model—since we assume that *honest providers* are able to ensure that at least a large fraction of their users base are genuine users faithfully following all Loopix protocols. Thus, the fraction of users controlled by the adversary may be capped to a small known fraction of the user base. We further assume that the adversary is able

to control a compromised user in a conversation with an honest user, and become a *conversation insider*.

An adversary is always assumed to have the GPA capability, but other capabilities depend on the adversary. We evaluate the security of Loopix in reference to these capabilities.

3.2.3 Security goals

The Loopix system aims to provide the following security properties against both passive and active attacks—including end-to-end correlation and $(n-1)$ attacks. These properties are inspired by the formal definitions from Anoa [2]. All security notions assume a strong adversary with information on all users, with up to one bit of uncertainty. In the following we write $\{S \to R\}$ to denote a communication from the sender S to the receiver R, $\{S \to\}$ to denote that there is a communication from S to any receiver and $\{S \not\to\}$ to denote that there is no communication from S to any receiver (S may still send cover messages). Analogously, we write $\{\to R\}$ to denote that there is a communication from any sender to the receiver R and $\{\not\to R\}$ to denote that there is no communication from any sender to R (however, R may still receive cover messages).

Sender-receiver third-party unlinkability. The senders and receivers should be unlinkable by any unauthorized party. Thus, we consider an adversary that wants to infer whether two users are communicating. We define *sender-receiver third party unlinkability* as the inability of the adversary to distinguish whether $\{S_1 \to R_1, S_2 \to R_2\}$ or $\{S_1 \to R_2, S_2 \to R_1\}$ for any concurrently online honest senders S_1, S_2 and honest receivers R_1, R_2 of the adversary's choice.

Loopix provides strong sender-receiver third-party anonymity against the GPA even in collaboration with corrupt mix nodes. We refer to Section 3.4.1 for our analysis of the unlinkability provided by individual mix nodes, to Section 3.4.3 for a quantitative analysis of the sender-receiver third-party anonymity of Loopix against the GPA and honest-but-curious mix nodes and to Section 3.4.2 for our discussion on active attacks.

Sender online unobservability. Whether or not senders are communicating should be hidden from an unauthorized party. We define *sender online unobservability* as the inability of an adversary to decide whether a specific sender S is communicating with any receiver $\{S \to\}$ or not $\{S \not\to\}$, for any concurrently online honest sender S of the adversary's choice.

Loopix provides strong sender online unobservability against the GPA in collaboration with an *insider* and even against a *corrupt provider*. We refer to Section 3.4.1 for our analysis of the latter.

Note, that sender online unobservability directly implies the notion of *sender anonymity* where the adversary tries to distinguish between two possible senders communicating with a target receiver. Formally, $\{S_1 \to R, S_2 \not\to\}$ or $\{S_1 \not\to, S_2 \to R\}$ for any concurrently online honest senders S_1 and S_2 and any receiver of the adversary's choice. Loopix provides sender anonymity even in light of a conversation insider, i.e., against a corrupt receiver.

Receiver unobservability. Whether or not receivers are part of a communication should be hidden from an unauthorized party. We define *receiver unobservability* as the inability of an adversary to decide whether there is a communication from any sender to a specific receiver R $\{\to R\}$ or not $\{\not\to R\}$, for any online or offline honest receiver R of the adversary's choice.

Loopix provides strong receiver unobservability against the GPA in collaboration with an *insider*, under the condition of an *honest provider*. We show in Section 3.4.1 how an honest provider assists the receiver in hiding received messages from third party observers.

Note, that receiver unobservability directly implies the notion of *receiver anonymity* where the adversary tries to distinguish between two possible receivers in communication with a target sender.

Symbol	Description
N	Mix nodes
P	Providers
λ_L	Loop traffic rate (user)
λ_D	Drop cover traffic rate (user)
λ_P	Payload traffic rate (user)
l	Path length (user)
μ	The mean delay at mix M_i
λ_M	Loop traffic rate (mix)

Table 3.1: Summary of notation

Formally, $\{S \to R_1, \not\to R_2\}$ or $\{\not\to R_1, S \to R_2\}$ for any concurrently online honest sender S and any two honest receivers R_1, R_2 of the adversary's choice. [2]

Non-goals. Loopix provides anonymous unreliable datagram transmission, as well as facilities the reply of sent messages (through add-ons). This choice allows for flexible traffic management, cover traffic, and traffic shaping. On the downside, higher-level applications using Loopix need to take care of reliable end-to-end transmission and session management. We leave the detailed study of those mechanisms as future work.

The provider based architecture supported by Loopix aims to enable managed access to the network, support anonymous blacklisting to combat abuse [26], and payments for differential access to the network [1]. However, we do not discuss these aspects of Loopix in this work, and concentrate instead on the core anonymity features and security properties described above.

3.3 The Loopix architecture

In this section we describe the Loopix system in detail—Figure 3.1 provides an overview. We also introduce the notation used further in the paper, summarized in Section 3.3.

3.3.1 System setup

The Loopix system consists of a set of mix nodes N and providers P. We consider a population of U users communicating through Loopix, each of which can act as *sender* and *receiver*, denoted by indices S_i, R_i, where $i \in \{1, \ldots, U\}$ respectively. Each entity of the Loopix infrastructure has its unique public-private key pair (sk, pk). In order for a *sender* S_i, with a key pair (sk_{S_i}, pk_{S_i}), to send a message to a *receiver* R_j, with a key pair (sk_{R_j}, pk_{R_j}), the sender needs to know the receiver's Loopix *network location*, i.e., the IP address of the user's provider and an identifier of the user, as well as the public encryption key pk_{R_j}. We assume this information can be made available through a privacy-friendly lookup or introduction system for initiating secure connection [31]. This is out of scope for this work.

3.3.2 Format, paths and cover traffic

Message packet format. All messages are *end-to-end encrypted* and encapsulated into packets to be processed by the mix network. We use the Sphinx packet design [15], to ensure that in-

[2] If the receiver's provider is honest, Loopix provides a form of receiver anonymity even in light of a conversation insider: a corrupt sender that only knows the pseudonym of a receiver cannot learn which honest client of a provider is behind the pseudonym.

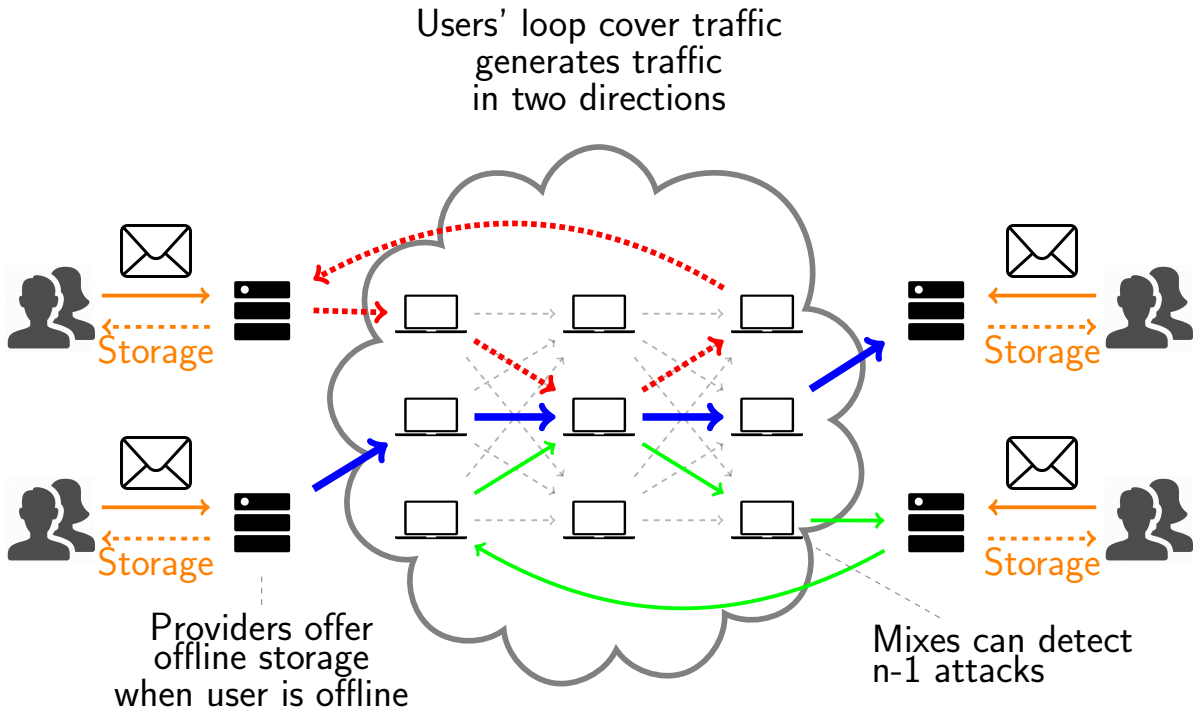

Figure 3.1: The Loopix Architecture. Clients pass the messages to the providers, which are responsible for injecting traffic into the network. The received messages are stored in individual inboxes and retrieved by clients when they are online.

termediate mixes learn no additional information beyond some routing information. All messages are padded to the same length, which hides the path length and the relay position and guarantees unlinkability at each hop of the messages' journey over the network. Each message wrapped into the Sphinx packet consists of two separate parts: a header H, carrying the layered encryption of meta-data for each hop, and the encrypted payload ρ of the message. The header provides each mix server on the path with confidential meta-data. The meta-data includes a single element of a cyclic group (used to derive a shared encryption/decryption key), the routing information and the message authentication code. We extend the Sphinx packet format to carry additional routing information in the header to each intermediate relay, including a delay and additional flags.

Path selection. As opposed to onion routing, in Loopix the communication path for every single message is chosen independently, even between the same pair of users.

Messages are routed through l layers of mix nodes, assembled in a stratified topology [11, 20]. Each mix node is connected only with all the mix nodes from adjacent layers. This ensures that few links are used, and those few links are well covered in traffic; stratified topologies mix well in few steps [20]. Providers act as the first and last layer of mix servers. To send a message, the sender encapsulates the routing information described above into a Sphinx packet which travels through their provider, a sequence of mix servers, until it reaches the receiver's provider and finally the *receiver*. For each of those hops the sender samples a delay from an exponential distribution with parameter μ, and includes it in the routing information to the corresponding relay.

Sending messages and cover traffic. Users and mix servers continuously generate a bed of *real* and *cover traffic* that is injected into the network. Our design guarantees that all outgoing traffic sent by users can by modeled by a Poisson process.

To send a message, a user packages their message into a mix packet and places it into their *buffer*—a first-in-first-out (FIFO) queue that stores all the messages scheduled to be sent.

Each sender periodically checks, following the exponential distribution with parameter $\frac{1}{\lambda_P}$, whether there is any scheduled message to be sent in their buffer. If there is a scheduled message, the sender pops this message from the buffer queue and sends it, otherwise a *drop* cover message is generated (in the same manner as a regular message) and sent (depicted as the the four middle blue arrows in Figure 3.1). Cover messages are routed through the sender's provider and a chain of mix nodes to a random destination provider. The destination provider detects the message is cover based on the special drop flag encapsulated into the packet header, and drops it. Thus, regardless of whether a user actually wants to send a message or not, there is always a stream of messages being sent according to a Poisson process $Pois(\lambda_P)$.

Moreover, independently from the above, all users emit separate streams of special indistinguishable types of *cover messages*, which also follow a Poisson process. The first type of cover messages are Poisson distributed *loops* emitted at rate λ_L. These are routed through the network and *looped back* to the senders (the upper four red arrows in Figure 3.1), by specifying the sending user as the recipient. These *"loops"* inspire the system's name. Users also inject a separate stream of *drop* cover messages, defined before, following the Poisson distribution $Pois(\lambda_D)$. Additionally, each user sends at constant time a stream of *pull* requests to its *provider* in order to retrieve received messages, described in Section 3.3.2.

Each mix also injects their own *loop* cover traffic, drawn from a Poisson process with rate $Pois(\lambda_M)$, into the network. Mix servers inject mix packets that are looped through a path, made up of a subset of other mix servers and one randomly selected *provider*, back to the sending mix server, creating a second type of *"loop"*. This loop originates and ends in a mix server (shown as the lower four green arrows in Figure 3.1). In Section 3.4 we examine how the *loops* and the *drop* cover messages help protect against passive and active attacks.

Message storing and retrieving. Providers do not forward the incoming mix packets to users but instead buffer them. Users, when online, *poll* providers or register their online status to download a fixed subset of stored messages, allowing for the reception of the off-line messages. Recall that cover loops are generated by users and traverse through the network and come back to the sender. Cover loops serve as a cover set of *outgoing* and *incoming* real messages. Whenever a user requests messages, their provider responds with a constant number of messages, which includes their cover loop messages and real messages. If the inbox of a particular user contains fewer messages than this constant number, the provider sends dummy messages to the sender up to that number.

3.3.3 The Poisson mix strategy

Loopix leverages cover traffic to resist traffic analysis while still achieving low- to mid-latency. To this end Loopix employs a mixing strategy that we call a *Poisson Mix*, to foil observers from learning about the correspondences between input and output messages. The Poisson Mix is a simplification of the Stop-and-go mix strategy [28]. A similar strategy has been used to model traffic in onion routing servers [13]. In contrast, recall that in Loopix each message is source routed through an independent route in the network.

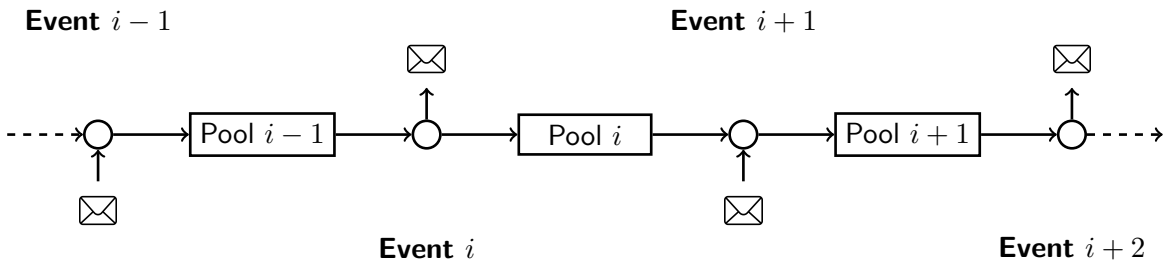

Figure 3.2: The Poisson Mix strategy mapped to a Pool mix strategy. Each single message sending or receiving event leads to a new pool of messages that are exchangeable and indistinguishable with respect to their departure times.

The Poisson Mix functions as follows: mix servers listen for the incoming mix packets and received messages are checked for duplication and decoded using the mix node's private keys. The detected duplicates are dropped. Next, the mix node extracts a subsequent mix packet. Decoded mix packets are not forwarded immediately, but each of them is delayed according to a source pre-determined delay d_i. Honest clients chose those delays, independently for each hop, from an exponential distribution with a parameter μ. We assume that this parameter is public and the same for all mix nodes.

Mathematical model of a Poisson mix. Honest clients and mixes generate drop cover traffic, loop traffic, and messaging traffic following a Poisson process. Aggregating Poisson processes results in a Poisson process with the sum of their rates, therefore we may model the streams of traffic received by a Poisson mix as a Poisson process. It is the superposition of traffic streams from multiple clients. It has a rate λ_n depending on the number of clients and the number of mix nodes.

Since this input process is a Poisson process and each message is independently delayed using an exponential distribution with parameter μ, the Poisson Mix may be modeled as an $M/M/\infty$ queuing system – for we have a number of well known theorems [4]. We know that output stream of messages is also a Poisson process with the parameter λ_n as the the input process. We can also derive the distribution of the number of messages within the Poisson Mix [33]:

Lemma 1. *The mean number of messages in the Poisson Mix with input Poisson process* $Pois(\lambda)$ *and exponential delay parameter* μ *at a steady state follows the Poisson distribution* $Pois\,(\lambda/\mu)$.

Those characteristics give the Poisson Mix its name. This allows us to calculate the mean number of messages *perfectly* mixed together at any time, as well as the probability that the number of messages falls below or above certain thresholds.

The Poisson Mix, under the assumption that it approximates an $M/M/\infty$ queue is a stochastic variant of a pool mixing strategy [38]. Conceptually, each message sending or receiving leads to a pool within which messages are indistinguishable due to the memoryless property of the exponential delay distribution.

Lemma 2 (Memoryless property [33])**.** *For an exponential random variable* X *with parameter* μ *holds* $\Pr[X > s + t | X > t] = \Pr[X > s]$.

Intuitively, any two messages in the same pool are emitted next with equal probability – no matter how long they have been waiting. As illustrated in Figure 3.2, the receiving event $i-1$ leads to a pool of messages $i-1$, until the sending event i. From the perspective of the adversary observing all inputs and outputs, all messages in the pool $i-1$ are indistinguishable from each other. Only the presence of those messages in the pool is necessary to characterize the hidden state of the mix (not their delay so far). Relating the Poisson mix to a pool mix allows us to compute easily and exactly both the entropy metric for the anonymity it provides [36] within a trace (used in Section 3.4.1). It also allows us to compute the likelihood that an emitted message was any specific input message used in our security evaluation.

3.4 Analysis of Loopix security properties

In this section we present the analytical and experimental evaluation of the security of Loopix and argue its resistance to traffic analysis and active attacks.

3.4.1 Passive attack resistance

Message indistinguishability

Loopix relies on the Sphinx packet format [15] to provide bitwise unlinkability of incoming and outgoing messages from a mix server; it does not leak information about the number of hops a single message has traversed or the total path length; and it is resistant to tagging attacks.

For Loopix, we make minor modifications to Sphinx to allow auxiliary meta-information to be passed to different mix servers. Since all the auxiliary information is encapsulated into the header of the packet in the same manner as any meta-information was encapsulated in the Sphinx design, the security properties are unchanged. An external adversary and a corrupt intermediate mix node or a corrupt provider will not be able to distinguish *real* messages from *cover* messages of any type. Thus, the GPA observing the network cannot infer any information about the type of the transmitted messages, and intermediate nodes cannot distinguish real messages, drop cover messages or loops of clients and other nodes from each other. Providers are able to distinguish *drop* cover message destined for them from other messages, since they learn the *drop flag* attached in the header of the packet. Each mix node learns the delay chosen by clients for this particular mix node, but all delays are chosen independently from each other.

Client-provider unobservability

In this section, we argue the *sender and receiver unobservability* against different adversaries in our threat model. Users emit payload messages following a Poisson distribution with parameter λ_P. All messages scheduled for sending by the user are placed within a first-in-first-out buffer. According to a Poisson process, a single message is popped out of the buffer and sent, or a drop cover message is sent in case the buffer is empty. Thus, from an adversarial perspective, there is always traffic emitted modeled by $Pois(\lambda_P)$. Since clients send also streams of cover traffic messages with rates λ_L for loops and λ_D for drop cover messages, the traffic sent by the client follows $Pois(\lambda_P + \lambda_L + \lambda_D)$. Thus we achieve perfect *sender unobservability*, since the adversary cannot tell whether a genuine message or a drop cover message is sent.

When clients query providers for received messages, the providers always send a constant number of messages to the client. If the number of messages in client's inbox is smaller than a constant threshold, the provider generates additional dummy messages. Thus, the adversary observing

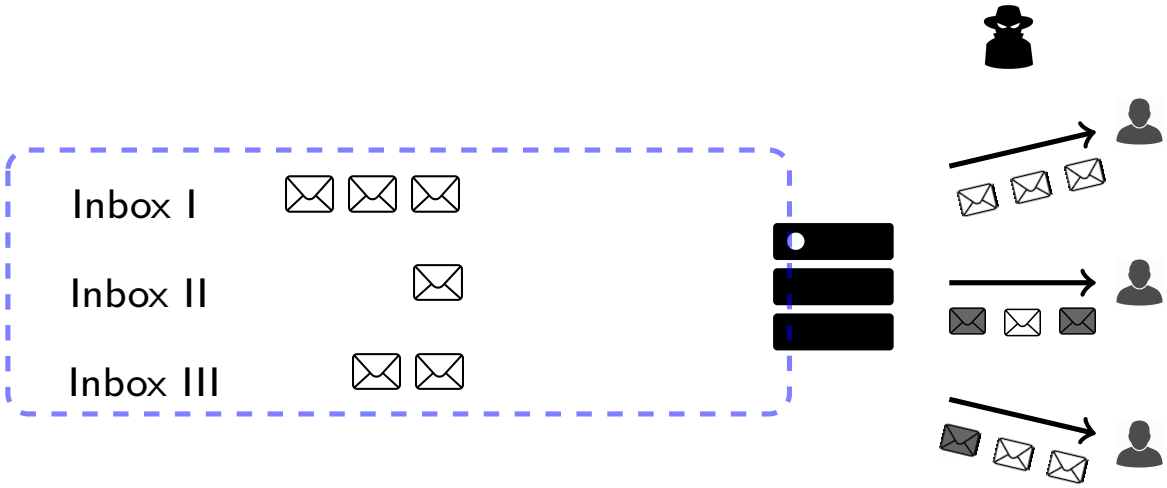

Figure 3.3: Provider stores messages destined for assigned clients in a particular inbox. When users pull messages from the mix node, the provider generates cover messages to guarantee that the adversary cannot learn how many messages are in the users inbox. The messages from the inbox and dummies are indistinguishable.

the client-provider connection, as presented on Figure 3.3, cannot learn how many messages were in the user's inbox. Note that, as long as the providers are honest, the protection and *receiver unobservability* is perfect and the adversary cannot learn any information about the inbox and outbox of any client.

If the provider is dishonest, then they are still uncertain whether a received message is genuine or the result of a client loop – something that cannot be determined from their bit pattern alone. However, further statistical attacks may be possible, and we leave quantifying exact security against this threat model as future work.

Poisson mix security

We first show that a single honest Poisson mix provides a measure of *sender-receiver unlinkability*. From the properties of Poisson mix, we know that the number of messages in the mix server at a steady state depends on the ratio of the incoming traffic (λ) and the delay parameter (μ) (from Section 3.3.3). The number of messages in each mix node at any time will on average be $\frac{\lambda}{\mu}$. However, an adversary observing the messages flowing into and out of a single mix node could estimate the exact number of messages within a mix with better accuracy – hindered only by the mix loop cover traffic.

We first consider, conservatively, the case where a mix node is not generating any loops and the adversary can count the exact number of messages in the mix. Let us define $o_{n,k,l}$ as an adversary A observing a mix in which n messages arrive and are mixed together. The adversary then observes an outgoing set of $n-k$ messages and can infer that there are now $k < n$ messages in the mix. Next, l additional messages arrive at the mix before any message leaves, and the pool now mixes $k+l$ messages. The adversary then observes exactly one outgoing message m and tries to correlate it with any of the $n+l$ messages which she has observed arriving at the mix node.

The following lemma is based on the memoryless property of the Poisson mix. It provides an upper bound on the probability that the adversary A correctly links the outgoing message m with one of the previously observed arrivals in observation $o_{n,k,l}$.

Theorem 1. *Let m_1 be any of the initial n messages in the mix node in scenario $o_{n,k,l}$, and let m_2 be any of the l messages that arrive later. Then*

$$Pr(m = m_1) = \frac{k}{n(l+k)}, \tag{3.1}$$

$$Pr(m = m_2) = \frac{1}{l+k}. \tag{3.2}$$

Note that the last l messages that arrived at the mix node have equal probabilities of being the outgoing message m, independently of their arrival times. Thus, the arrival and departure times of the messages cannot be correlated, and the adversary learns no additional information by observing the timings. Note that $\frac{1}{l+k}$ is an upper bound on the probability that the adversary A correctly links the outgoing message to an incoming message. Thus, continuous observation of a Poisson mix leaks no additional information other than the number messages present in the mix. We leverage those results from about a single Poisson Mix to simulate the information propagated withing a the whole network observed by the adversary (c.f. Section 3.4.3).

We quantify the anonymity of messages in the mix node empirically, using an information theory based metric introduced in [36, 17]. We record the traffic flow for a single mix node and compute the distribution of probabilities that the outgoing message is the adversary's target message. Given this distribution we compute the value of Shannon entropy, a measure of unlinkability of incoming to outgoing messages. We compute this using the simpy package in Python. All data points are averaged over 50 simulations.

Figure 3.4 depicts the change of entropy against an increasing rate of incoming mix traffic λ. We simulate the dependency between entropy and traffic rate for different mix delay parameter μ by recording the traffic flow and changing state of the mix node's pool. As expected, we observe that for a fixed delay, the entropy increases when the rate of traffic increases. Higher delay also results in an increase in entropy, denoting a larger potential anonymity set, since more messages are mixed together.

In case the mix node emits loop cover traffic, the adversary with observation $o_{n,k,l}$, tries to estimate the probability that the observed outgoing message is a particular *target* message she observed coming into the mix node. An outgoing message can be either input message or a loop message generated by the mix node – resulting in additional uncertainty for the adversary.

Theorem 2. *Let m_1 be any of the initial n messages in the mix node in scenario $o_{n,k,l}$, and let m_2 be any of the l messages that arrive later. Let λ_M denote the rate at which mix node generates loop cover traffic. Then,*

$$\Pr(m = m_2) = \frac{k}{n} \cdot \frac{\mu}{(l+k)\mu + \lambda_M},$$
$$\Pr(m = m_1) = \frac{\mu}{(l+k)\mu + \lambda_M}.$$

We conclude that the loops generated by the mix node obfuscate the adversary's view and decrease the probability of successfully linking input and output of the mix node. In Section 3.4.2 we show that those types of loops also protect against active attacks.

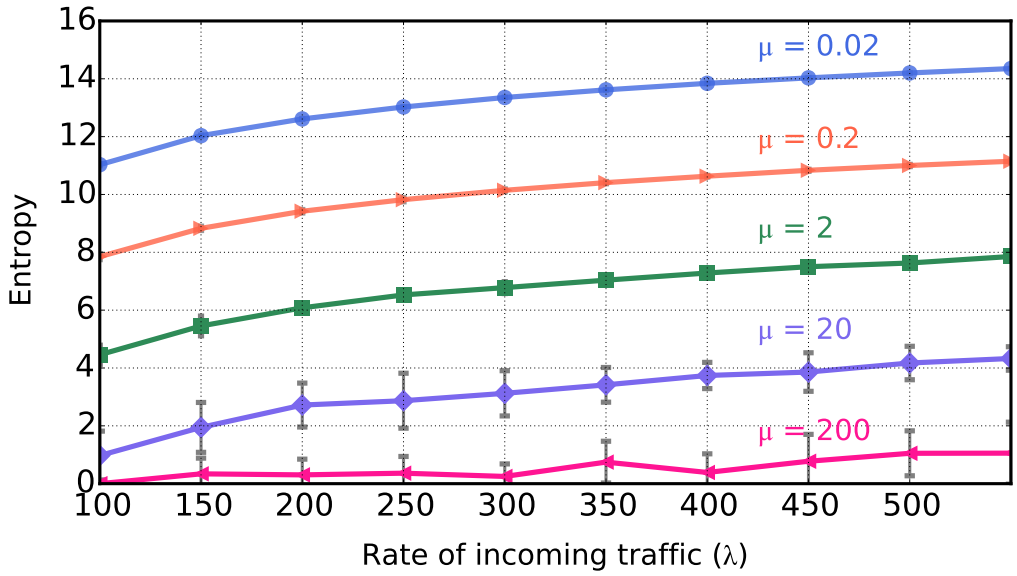

Figure 3.4: Entropy versus the changing rate of the incoming traffic for different delays with mean $\frac{1}{\mu}$. In order to measure the entropy we run a simulation of traffic arriving at a single Loopix mix node.

3.4.2 Active-attack resistance

Lemma 1 gives the direct relationship between the expected number of messages in a mix node, the rate of incoming traffic, and the delay induced on a message while transiting through a mix. By increasing the rate of cover traffic, λ_D and λ_L, users can collectively maintain strong anonymity with low message delay. However, once the volume of real communication traffic λ_P increases, users can tune down the rate of cover traffic in comparison to the real traffic, while maintaining a small delay and be confident their messages are mixed with a sufficient number of messages.

In the previous section, we analyze the security properties of Loopix when the adversary observes the state of a single mix node and the traffic flowing through it. We showed, that the adversary's advantage is bounded due to the indistinguishability of messages and the memoryless property of the Poisson mixing strategy. We now investigate how Loopix can protect users' communications against active adversaries conducting the $(n-1)$ attack.

Active attacks

We consider an attack at a mix node where an adversary blocks all but a target message from entering in order to follow the target message when it exits the mix node. This is referred to as an *(n-1) attack* [37].

A mix node needs to distinguish between an active attack and loop messages dropped due to congestion. We assume that each mix node chooses some public parameter r, which is a fraction of the number of loops that are expected to return. If the mix node does not see this fraction of loops returning they alter their behavior. In extremis such a mix could refuse to emit any messages – but this would escalate this attack to full denial-of-service. A gentler approach involves generating

more cover traffic on outgoing links [16].

To attempt an *(n-1) attack*, the adversary could simply block all incoming messages to the mix node except for a target message. The Loopix mix node can notice that the self-loops are not returning and deduce it is under attack. Therefore, an adversary that wants to perform a stealthy attack has to be judicious when blocking messages, to ensure that a fraction r of loops return to the mix node, i.e. the adversary must distinguish loop cover traffic from other types of traffic. However, traffic generated by mix loops is indistinguishable from other network traffic and they cannot do this better than by chance. Therefore given a threshold $r = \frac{\lambda_M}{s}, s \in \mathbb{R}_{>1}$ of expected returning loops when a mix observes fewer returning it deploys appropriate countermeasures.

We analyze this strategy: since the adversary cannot distinguish loops from other traffic the adversary can do no better than block traffic uniformly such that a fraction $R = \frac{\lambda}{s} = \frac{\lambda_R + \lambda_M}{s}$ enter the mix, where λ_R is the rate of incoming traffic that is not the mix node's loops. If we assume a steady state, the target message can expect to be mixed with $\frac{\lambda_R}{s \cdot \mu}$ messages that entered this mix, and $\frac{\lambda_M}{\mu}$ loop messages generated at the mix node. Thus, the probability of correctly blocking a sufficient number of messages entering the mix node so as not to alter the behavior of the mix is:

$$\Pr(x = \text{target}) = \frac{1}{\lambda_R/s \cdot \mu + \lambda_M/\mu} = \frac{s\mu}{s\lambda_M + \lambda_R}$$

Due to the stratified topology, providers are able to distinguish mix loop messages sent from other traffic, since they are unique in not being routed to or from a client. This is not a substantial attack vector since mix loop messages are evenly distributed among all providers, of which a small fraction are corrupt and providers do not learn which mix node sent the loop to target it.

3.4.3 End-to-end anonymity evaluation

We evaluate the *sender-receiver third-party unlinkability* of the full Loopix system through an empirical analysis of the propagation of messages in the network. Our key metric is the *expected difference in likelihood* that a message leaving the last mix node is sent from one sender in comparison to another sender. Given two probabilities $p_0 = \Pr[S_0]$ and $p_1 = \Pr[S_1]$ that the message was sent by senders S_0 and S_1, respectively, we calculate

$$\varepsilon = \left|\log\left(p_0/p_1\right)\right|. \tag{3.3}$$

To approximate the probabilities p_0 and p_1, we proceed as follows. We simulate $U = 100$ senders that generate and send messages (both payload and cover messages) with a rate $\lambda = 2$. Among them are two challenge senders S_0 and S_1 that send payload messages at a constant rate, i.e, they add one messages to their sending buffer every time unit.

Whenever a challenge sender S_0 or S_1 sends a payload message from its buffer, we tag the message with a label S_0 or S_1, respectively. All other messages, including messages from the remaining 98 clients and the cover messages of S_0 and S_1 are unlabeled. At every mix we track the probability that an outgoing message is labeled S_0 or S_1, depending on the messages that entered the mix node and the number of messages that already left the mix node, as in Theorem 1. Thus, messages leaving a mix node carry a probability distribution over labels S_0, S_1, or 'unlabeled'. Corrupt mix nodes, assign to outgoing messages their input distributions. The probabilities naturally add up to 1. For example, a message leaving a mix can be labeled as $\{S_0 : 12\%, S_1 : 15\%, \text{unlabeled} : 73\%\}$.

In a burn-in phase of 2500 time units, the 98 senders without S_0 or S_1 communicate. Then we start the two challenge senders and then simulate the network for another 100 time units, before we compute the *expected difference in likelihood* metric.

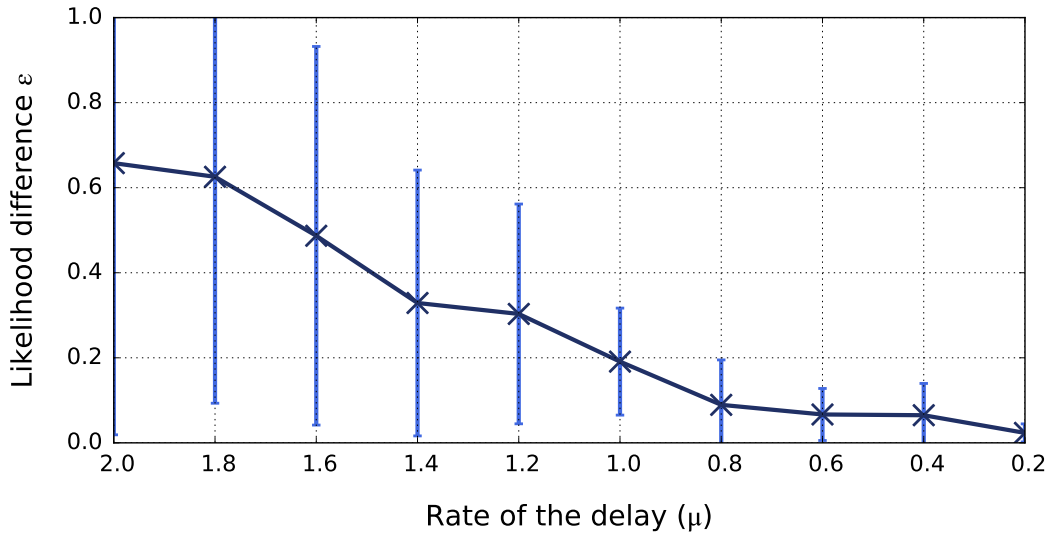

Figure 3.5: Likelihood difference ε depending on the delay parameter μ of mix nodes. We use $\lambda = 2$, a topology of 3 layers with 3 nodes per layer and no corruption.

This is a conservative approximation: we tell the adversary which of the messages leaving senders S_0 and S_1 are payload messages; and we do not consider mix or client loop messages confusing them. [3] However, when we calculate our anonymity metric at a mix node we assume this mix node to be honest.

Results

We compare our metric for different parameters: depending on the delay parameter μ, the number of layers in our topology l and the percentage of corrupted mix nodes in the network. All of the below simulations are averaged over 100 repetitions and the error bars are defined by the standard deviation.

Delay. Increasing the average delay (by decreasing parameter μ) with respect to the rate of message sending λ immediately increases anonymity (decreases ε) (Figure 3.5). For $\mu = 2.0$ and $\lambda/\mu = 1$, Loopix still provides a weak form of anonymity. As this fraction increases, the log likelihood ratio grow closer and closer to zero. We consider values $\lambda/\mu \geq 2$ to be a good choice in terms of anonymity.

Number of layers. By increasing the number of layers of mix nodes, we can further strengthen the anonymity of Loopix users. As expected, using only one or two layers of mix nodes leads to high values of adversary advantage ε. An increasing number of layers ε approaches zero (Figure 3.6). We consider a number of 3 or more layers to be a good choice. We believe the bump between 5–8

[3] The soundness of our simplification can be seen by the fact that we could tell the adversary which messages are loops and the adversary could thus ignore them. This is equivalent to removing them, as an adversary could also simulate loop messages.

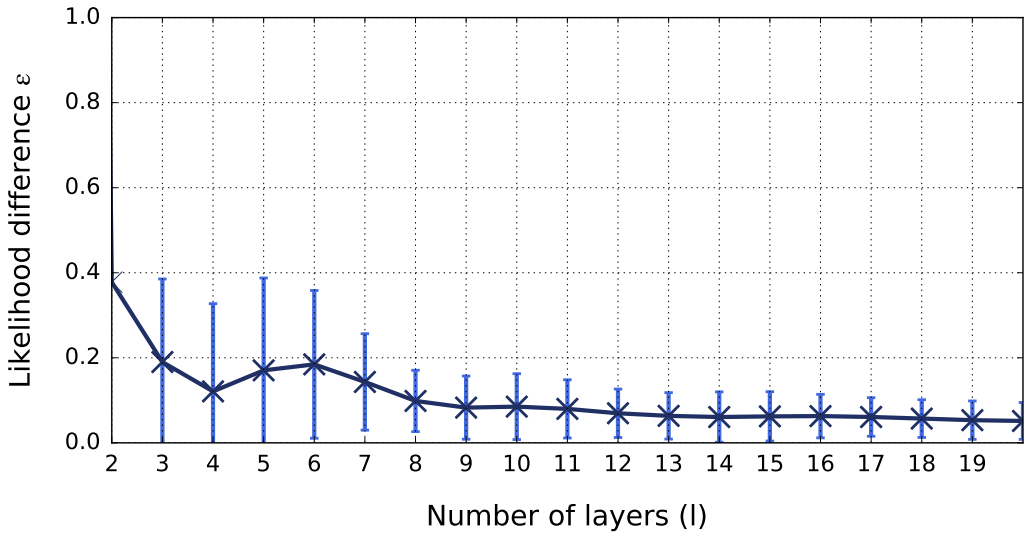

Figure 3.6: Likelihood difference ε depending on the number of layers of mix nodes with 3 mix nodes per layer. We use $\lambda = 2$, $\mu = 1$, and no corruption.

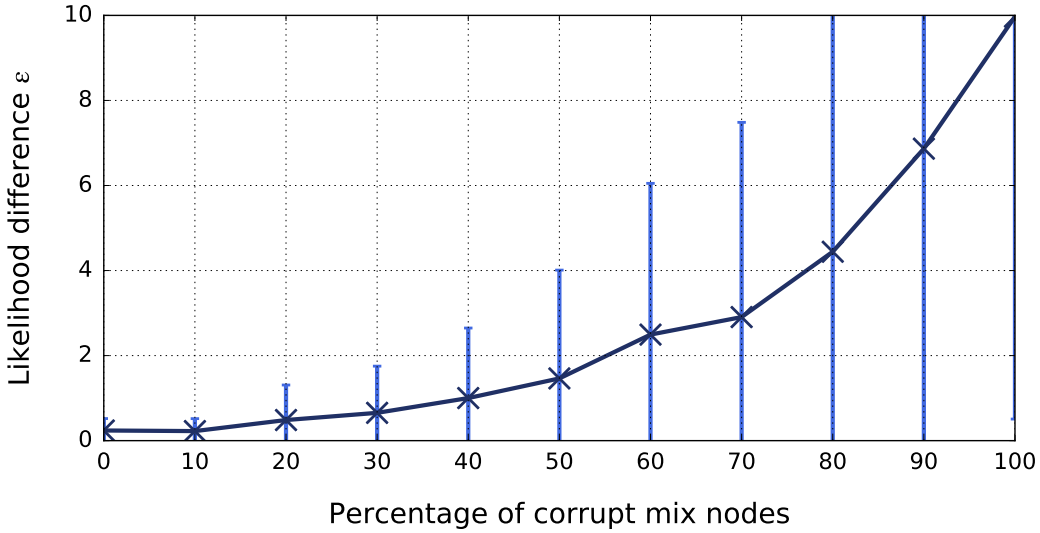

Figure 3.7: Likelihood difference ε depending on the percentage of (passively) corrupted mix nodes. We use $\lambda = 2$, $\mu = 1$ and a topology of 3 layers with 3 nodes per layer.

layers is due to messages not reaching latter layers within 100 time units, however results from experiments with increased duration do not display such a bump.

Corruption. Finally, we analyze the impact that corrupt mix nodes have on the adversary advantage ε (Figure 3.7). We assume that the adversary randomly corrupts mix nodes. Naturally,

the advantage ε increases with the percentage of corrupt mix nodes in the network. In a real-world deployment we do not expect a large fraction of mix nodes to be corrupt. While the adversary may be able to observe the entire network, to control a large number of nodes would be more costly.

3.5 Performance evaluation

Implementation. We implement the Loopix system prototype in 4000 lines of Python 2.7 code for *mix nodes*, *providers* and *clients*, including unit-tests, deployment, and orchestration code. Loopix source code is available under an open-source license[4]. We use the Twisted 15.5.0 network library for networking; as well as the Sphinx mix packet format[5] and the cryptographic tools from the *petlib*[6] library. We modify Sphinx to use NIST/SEGS-p224 curves and to accommodate additional information inside the packet, including the delay for each hop and auxiliary flags. We also optimize its implementation leading to processing times per packet of less than $1ms$.

The most computationally expensive part of Loopix is messages processing and packaging, which involves cryptographic operations. Thus, we implement Loopix as a multi-thread system, with cryptographic processing happening in a thread pool separated from the rest of the operations in the main thread loop. To recover from congestion we implement active queue management based on a PID controller and we drop messages when the size of the queue reaches a (high) threshold.

Experimental setup. We present an experimental performance evaluation of the Loopix system running on the `AWS EC2` platform. All mix nodes and providers run as separate instances. Mix nodes are deployed on `m4.4xlarge` instances running `EC2 Linux` on $2.3\,GHz$ machines with $64\,GB$ RAM memory. Providers, since they handle more traffic, storage and operations, are deployed on `m4.16xlarge` instances with $256\,GB$ RAM. We select large instances to ensure that the providers are not the bottleneck of the bandwidth transfer, even when users send messages at a high rate. This reflects real-world deployments where providers are expected to be well-resourced. We also run one `m4.16xlarge` instance supporting 500 clients. We highlight that each client runs as a separate process and uses a unique port for transporting packets. Thus, our performance measurements are obtained by simulating a running system with independent clients[7]. In order to measure the system performance, we run six mix nodes, arranged in a stratified topology with three layers, each layer composed of two mix nodes. Additionally, we run four providers, each serving approximately 125 clients. The delays of all the messages are drawn from an exponential distribution with parameter μ, which is the same for all mix servers in the network. All measurements are take from network traffic dumps using `tcpdump`.

Bandwidth. First, we evaluate the maximum bandwidth of mix nodes by measuring the rate at which a single mix node processes messages, for an increasing overall rate at which users send messages.

We set up the fixed delay parameter $\mu = 1000$ (s.t. the average delay is $1\,ms$). We have 500 clients actively sending messages at rate λ each, which is the sum of payload, loop and drop rates, i.e., $\lambda = Pois(\lambda_L + \lambda_D + \lambda_P)$. We start our simulation with parameters $\lambda_L = \lambda_D = 1$ and $\lambda_P = 3$

[4]Public Github repository URL obscured for review.
[5]http://sphinxmix.readthedocs.io/en/latest/
[6]http://petlib.readthedocs.org
[7]Other works, e.g. [39, 40], report impressive evaluations in terms of scale, but in fact are simple extrapolations and not based on empirical results.

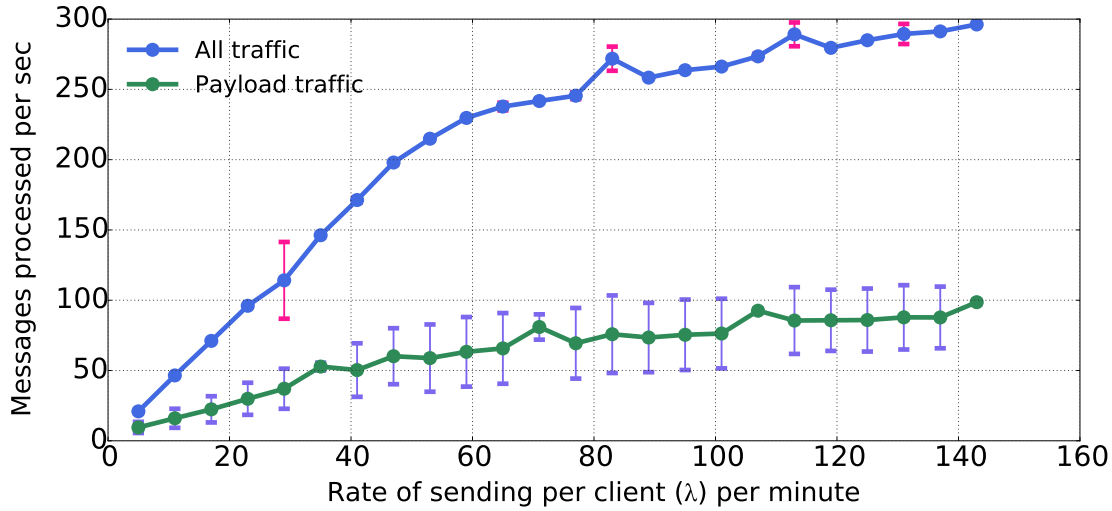

Figure 3.8: Overall bandwidth and good throughput per second for a single mix node.

messages per minute for a single client. Mix nodes send loop cover traffic at rate starting from $\lambda_M = 1$. Next, we periodically increase each Poisson rate by another 2 messages per minute.

In order to measure the overall bandwidth, i.e. the number of all messages processed by a single mix node, we use the network packet analyzer `tcpdump`. Since real and cover message packets are indistinguishable, we measure the good throughput by encapsulating an additional, temporary, *typeFlag* in the packet header for this evaluation, which leaks to the mix the message type—real or cover—and is recorded.

Figure 3.8 illustrates the number of total messages and the number of payload messages that are processed by a single mix node versus the overall sending rate λ of a single user. We observe that the bandwidth of the mix node increases linearly until it reaches around 225 messages per second. After that point the performance of the mix node stabilizes and we observe a much smaller growth. We highlight that the amount of *real* traffic in the network depends on the parameter λ_P within λ. A client may chose to tune up the rate of real messages sent, by tuning down the rate of loops and drop messages – at the potential loss of security in case less cover traffic is present in the system overall. Thus, depending on the size of the honest user population in Loopix, we can increase the rate of goodput.

Latency overhead & scalability. End-to-end latency overhead is the cost of routing and decoding relayed messages, without any additional artificial delays. We run simulations to measure its sensitivity to the number of users participating in the system.

We measure the time needed to process a single packet by a mix node, which is approximately $0.6\,ms$. This cost is dominated by the scalar multiplication of an elliptic curve point and symmetric cryptographic operations. For the end-to-end measurement, we run Loopix with a setup where all users have the same rates of sending real and cover messages, such that $\lambda_P = \lambda_D = \lambda_L = 10$ messages per minute and mix servers generate loops at rate $\lambda_M = 10$ messages per minute. All clients set a delay of 0.0 seconds for all the hops of their messages – to ensure we only measure the system overhead, not the artificial mixing delay.

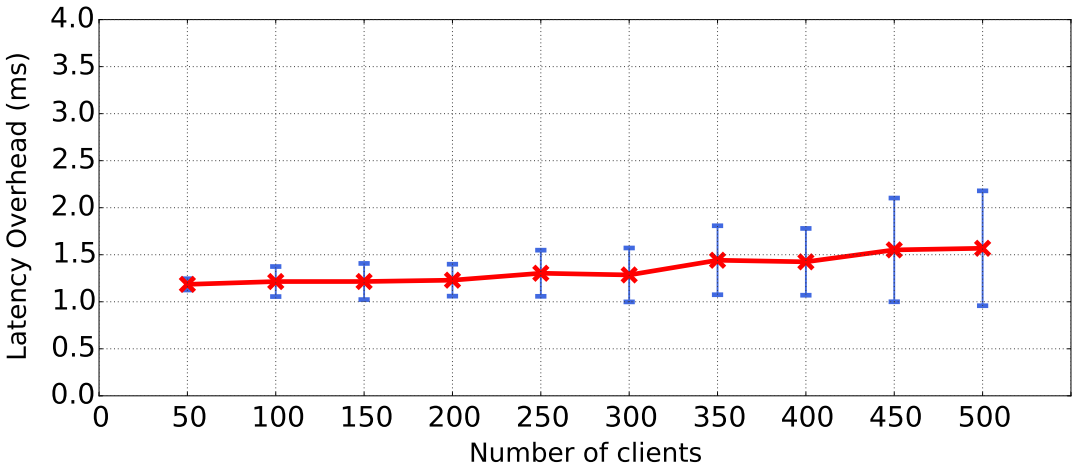

Figure 3.9: Latency overhead of the system where 50 to 500 users simultaneously send traffic at rates $\lambda_P = \lambda_L = \lambda_D = 10$ per minute and mix nodes generate loop cover traffic at rate $\lambda_M = 10$ per minute. We assume that there is not additional delay added to the messages by the senders.

Figure 3.9 shows that increasing the number of online clients, from 50 to 500, raises the latency overhead by only $0.37\,ms$. The variance of the processing delay increases with the amount of traffic in the network, but more clients do not significantly influence the average latency overhead. Neither the computational power of clients nor mix servers nor the communication between them seem to become bottlenecks for these rates. Those results show that the increasing number of users in the network does not lead to any bottleneck for our parameters. The measurements presented here are for a network of 6 mix nodes, however we can increase the system capacity by adding more servers. Thus, Loopix scales well for an increasing number of users.

We also investigate how increasing the delays through Poisson Mixing with $\mu = 2$ affects the end-to-end latency of messages. We measure this latency through timing mix heartbeat messages traversing the system. Figure 3.10 illustrates that when the mean delay $1/\mu$ sec. is higher than the processing time ($\sim 1\,ms - 2\,ms$), the end-to-end latency is determined by this delay, and follows the Gamma distribution with parameter being the sum of the exponential distribution parameter over the number of servers on the path. The good fit to a gamma distribution provides evidence that the implementation of Loopix is faithful to the queuing theory models our analysis assumes.

3.6 Related work

All anonymous communication designs share the common goal of hiding users' communication patterns from adversaries. Simultaneously minimizing latency and communication overhead while still providing high anonymity is challenging. We survey other anonymous systems and compare them with Loopix (a summary is provided in Table 3.2).

Early designs. Designs based on Chaum's mixes [7] can support both high and low latency communication; all sharing the basic principles of mixing and layered encryption. Mixmaster [34] supports sender anonymity using messages encryption but does not ensure receiver anonymity. Mixminion [14] uses fixed sized messages and supports anonymous replies and ensures forward

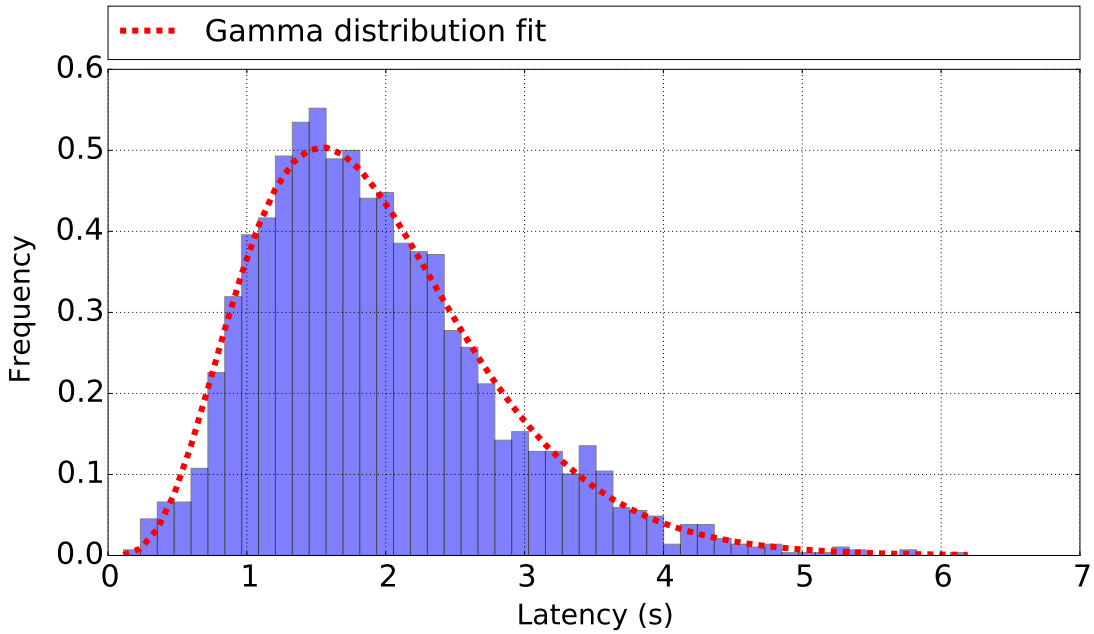

Figure 3.10: End-to-end latency histogram measured through timing mix node loops. We run 500 users actively communicating via Loopix at rates $\lambda_P = \lambda_L = \lambda_D = 60$ per minute and $\lambda_M = 60$ per minute. The delay for each hop is drawn from $Exp(2)$. The latency of the message is determined by the assigned delay and fits the Gamma distribution with mean 1.93 and standard deviation 0.87.

anonymity using link encryption between nodes. As a defense against traffic analysis, but at the cost of high-latencies, both designs delay incoming messages by collecting them in a pool that is flushed every t seconds (if a fixed message threshold is reached).

In contrast, Onion routing [25] was developed for low-latency anonymous communication. Similar to mix designs, each packet is encrypted in layers, and is decrypted by a chain of authorized onion routers. Tor [19], the most popular low-latency anonymity system, is an overlay network of onion routers. Tor protects against sender-receiver message linking against a partially global adversary and ensures perfect forward secrecy, integrity of the messages, and congestion control. However, Tor is vulnerable to traffic analysis attacks, if an adversary can observe the ingress and egress points of the network.

Recent designs. Vuvuzela [40] protects against both passive and active adversaries as long as there is one honest mix node. Since Vuvuzela operates in rounds, offline users lose the ability to receive messages and all messages must traverse a single chain of relay servers. Loopix does not operate in rounds, allows off-line users to receive messages and uses parallel mix nodes to improve the scalability of the network.

Stadium [39] and AnonPop [23] refine Vuvuzela; both operating in rounds making the routing of messages dependent on the dynamics of others. Stadium is scalable, but it lacks offline storage, whereas AnonPop does provide offline message storage. Loopix also provides both properties, and because it operates continuously avoids user synchronization issues.

Riposte [10] is based on a *write* PIR scheme in which users write their messages into a database, without revealing the row into which they wrote to the database server. Riposte enjoys low

	Low Latency	Low Communication Overhead	Scalable Deployment	Asynchronous Messaging†	Active Attack Resistant	Offline Storage*	Resistance to GPA
Loopix	✓	✓	✓	✓	✓	✓	✓
Dissent [41]	×	×	×	×	✓	×	✓
Vuvuzela [40]	×	×	✓	×	✓	×	✓
Stadium [39]	×	✓	✓	×	✓	×	✓
Riposte [10]	×	×	✓	×	✓	×	✓
Atom [29]	×	✓	✓	×	✓	×	✓
Riffle [30]	✓	✓	×	×	✓	×	✓
AnonPoP [23]	×	✓	✓	×	×	✓	✓
Tor [19]	✓	✓	✓	✓	×	×	×

Table 3.2: Comparison of popular anonymous communication systems. By *, we mean if the design intentionally incorporates provisions for delivery of messages when a user is offline, perhaps for a long period of time. By †, we mean that the system operates continuously and does not depend on synchronized rounds for its security properties and users do not need to coordinate to communicate together.

communication-overhead and protects against traffic analysis and denial of service attacks, but requires long epochs and a small number of clients writing into the database simultaneously. In contrast to Loopix, it is suitable for high-latency applications.

Dissent [8], based on DC-networks [8], offers resilience against a GPA and some active attacks, but at significantly higher delays and scales to only several thousand clients.

Riffle [30] introduces a new verifiable shuffle technique to achieve sender anonymity. Using PIR, Riffle guarantees receiver anonymity in the presence of an active adversary, as well as both sender and receiver anonymity, but it cannot support a large user base. Riffle also utilizes rounds protect traffic analysis attacks. Riffle is not designed for Internet-scale anonymous communication, like Loopix, but for supporting intra-group communication.

Finally, Atom [29] combines a number of novel techniques to provide mid-latency communication, strong protection against passive adversaries and uses zero knowledge proofs between servers to resist active attacks. Performance scales horizontally, however latency comparisons between Loopix and Atom are difficult due to the dependence on pre-computation in Atom. Unlike Loopix, Atom is designed for latency tolerant uni-directional anonymous communication applications with only sender anonymity in mind.

3.7 Discussion & future work

As shown in Section 3.4.1, the security of Loopix heavily depends on the ratio of the rate of traffic sent through the network and the mean delay at every mix node. Optimization of this ratio application dependent. For applications with small number of messages and delay tolerance, a small amount of cover traffic can guarantee security.

Loopix achieves its stated security and performance goals. However, there are many other facets of the design space that have been left for future work. For instance, reliable messages delivery, session management, and flow control while all avoiding inherent risks, such as statistical disclosure attacks [12], are all fruitful avenues of pursuit.

We also leave the analysis of replies to messages as future work. Loopix currently allows two methods if the receiver does not already know the sender a priori: we either attach the address of the sender to each message payload, or provide a single-use anonymous reply block [14, 15], which

enables different use-cases.

The Loopix architecture deliberately relies on established providers to connect to and authenticate end-users. This architecture brings a number of potential benefits, such as resistance to Sybil attacks, enabling anonymous blacklisting [26] and payment gateways [1] to mitigate flooding attacks and other abuses of the system, and privacy preserving measurements [22, 27] about client and network trends and the security stance of the system. All of this analysis is left for future work.

It is also apparent that an efficient and secure private lookup system, one that can deliver network state and keying information to its users, is necessary to support modern anonymous communications. Proposals of stand-alone 'presence' systems such as DP5 [5] and MP3 [35] provide efficient lookup methods, however, we anticipate that tight integration between the lookup and anonymity systems may bring mutual performance and security benefits, which is another avenue for future work.

3.8 Conclusion

The Loopix mix system explores the design space frontiers of low-latency mixing. We balance cover traffic and message delays to achieve a tunable trade-off between real traffic and cover traffic, and between latency and good anonymity. Low-latency incentivizes early adopters to use the system, as they benefit from good performance. Moreover, the cover traffic introduced by both clients and mix servers provides security in the presence of a smaller user-base size. In turn this promotes growth in the user-base leading on one hand to greater security [18], and on the other a tuning down of cover traffic over time.

Loopix is the first system to combine a number of best-of-breed techniques: we provide definitions inspired by AnoA [2] for our security properties; improve the analysis of simplified variants of stop-and-go-mixing as a Poisson mix [28]; we use restricted topologies to promote good mixing [20]; we deploy modern active attack mitigations based on loops [16]; and we use modified modern cryptographic packet formats, such as Sphinx [15], for low information leakage. Our design, security and performance analysis, and empirical evaluation shows they work well together to provide strong security guarantees.

The result of composing these different techniques – previously explored as separate and abstract design options – is a design that is strong against global network level adversaries without the very high-latencies traditionally associated with mix systems [34, 14]. Thus, Loopix revitalizes message-based mix systems and makes them competitive once more against onion routing [25] based solutions that have dominated the field of anonymity research since Tor [19] was proposed in 2004.

Bibliography

[1] E. Androulaki, M. Raykova, S. Srivatsan, A. Stavrou, and S. M. Bellovin. PAR: Payment for Anonymous Routing. In *Privacy Enhancing Technologies, 8th International Symposium, PETS 2008, Leuven, Belgium, July 23-25, 2008, Proceedings*, pages 219–236, 2008.

[2] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi. AnoA: A Framework for Analyzing Anonymous Communication Protocols. In *Computer Security Foundations Symposium (CSF), 2013 IEEE 26th*, pages 163–178. IEEE, 2013.

[3] H. Ballani, P. Francis, and X. Zhang. A study of prefix hijacking and interception in the Internet. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 265–276. ACM, 2007.

[4] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[5] N. Borisov, G. Danezis, and I. Goldberg. DP5: A private presence service. *Proceedings on Privacy Enhancing Technologies*, 2015(2):4–24, 2015.

[6] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 605–616. ACM, 2012.

[7] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[8] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[9] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig. HORNET: High-speed Onion Routing at the Network Layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1441–1454, 2015.

[10] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy*, pages 321–338. IEEE, 2015.

[11] G. Danezis. Mix-networks with restricted routes. In *International Workshop on Privacy Enhancing Technologies*, pages 1–17. Springer, 2003.

[12] G. Danezis. Statistical disclosure attacks. In *Security and Privacy in the Age of Uncertainty*, pages 421–426. Springer, 2003.

[13] G. Danezis. The Traffic Analysis of Continuous-Time Mixes. In *Privacy Enhancing Technologies, 4th International Workshop, PET 2004, Toronto, Canada, May 26-28, 2004, Revised Selected Papers*, pages 35–50, 2004.

[14] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Security and Privacy, 2003. Proceedings. 2003 Symposium on*, pages 2–15. IEEE, 2003.

[15] G. Danezis and I. Goldberg. Sphinx: A compact and provably secure mix format. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 269–282. IEEE, 2009.

[16] G. Danezis and L. Sassaman. Heartbeat traffic to counter (n-1) attacks: red-green-black mixes. In *Proceedings of the 2003 ACM workshop on Privacy in the electronic society*, pages 89–93. ACM, 2003.

[17] C. Diaz, S. Seys, J. Claessens, and B. Preneel. Towards measuring anonymity. In *International Workshop on Privacy Enhancing Technologies*, pages 54–68. Springer, 2002.

[18] R. Dingledine and N. Mathewson. Anonymity Loves Company: Usability and the Network Effect. In *WEIS*, 2006.

[19] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[20] R. Dingledine, V. Shmatikov, and P. Syverson. Synchronous batching: From cascades to free routes. In *International Workshop on Privacy Enhancing Technologies*, pages 186–206. Springer, 2004.

[21] J. R. Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

[22] T. Elahi, G. Danezis, and I. Goldberg. Privex: Private collection of traffic statistics for anonymous communication networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1068–1079. ACM, 2014.

[23] N. Gelernter, A. Herzberg, and H. Leibowitz. Two Cents for Strong Anonymity: The Anonymous Post-office Protocol. Cryptology ePrint Archive, Report 2016/489, 2016. `http://eprint.iacr.org/2016/489`.

[24] Y. Gilad and A. Herzberg. Spying in the dark: TCP and Tor traffic analysis. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 100–119. Springer, 2012.

[25] D. Goldschlag, M. Reed, and P. Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[26] R. Henry and I. Goldberg. Thinking inside the BLAC box: smarter protocols for faster anonymous blacklisting. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 71–82. ACM, 2013.

[27] R. Jansen and A. Johnson. Safely Measuring Tor. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1553–1567, 2016.

[28] D. Kesdogan, J. Egner, and R. Büschkes. Stop-and-go-mixes providing probabilistic anonymity in an open system. In *International Workshop on Information Hiding*, pages 83–98. Springer, 1998.

[29] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford. Atom: Scalable Anonymity Resistant to Traffic Analysis. *CoRR*, abs/1612.07841, 2016.

[30] Y. H. Kwon. *Riffle: An efficient communication system with strong anonymity*. PhD thesis, Massachusetts Institute of Technology, 2015.

[31] D. Lazar and N. Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *Proceedings of the 12th Symposium on Operating Systems Design and Implementation (OSDI), Savannah, GA*, 2016.

[32] S. Le Blond, D. Choffnes, W. Caldwell, P. Druschel, and N. Merritt. Herd: A Scalable, Traffic Analysis Resistant Anonymity Network for VoIP Systems. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 639–652. ACM, 2015.

[33] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

[34] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Protocol-Version 2. Draft. *July, available at: www. abditum. com/mixmaster-spec. txt*, 2003.

[35] R. Parhi, M. Schliep, and N. Hopper. MP3: A More Efficient Private Presence Protocol. *arXiv preprint arXiv:1609.02987*, 2016.

[36] A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In *International Workshop on Privacy Enhancing Technologies*, pages 41–53. Springer, 2002.

[37] A. Serjantov, R. Dingledine, and P. Syverson. From a trickle to a flood: Active attacks on several mix types. In *International Workshop on Information Hiding*, pages 36–52. Springer, 2002.

[38] A. Serjantov and R. E. Newman. On the anonymity of timed pool mixes. In *Security and Privacy in the Age of Uncertainty*, pages 427–434. Springer, 2003.

[39] N. Tyagi, Y. Gilad, M. Zaharia, and N. Zeldovich. Stadium: A Distributed Metadata-Private Messaging System. Cryptology ePrint Archive, Report 2016/943, 2016. `http://eprint.iacr.org/2016/943`.

[40] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152. ACM, 2015.

[41] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson. Dissent in numbers: Making strong anonymity scale. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 179–182, 2012.

4. Strong and robust security guarantees in the presence of malicious components

The semiconductor industry is fully globalized and integrated circuits (ICs) are commonly defined, designed and fabricated in different premises across the world. However, this exposes vendors to supply chain attacks, where insiders may introduce malicious circuitry in the final products. This is of particular concern to cryptographic hardware, storing and computing on confidential data. So far, all mitigation attempts have been either circumvented by new attacks in a short time, or come with unrealistically high manufacturing costs and complexity.

This paper proposes Myst, a practical high-assurance architecture, that uses commercial off-the-shelf (COTS) hardware, and provides strong security guarantees, even in the presence of multiple malicious components. The key idea is to combine protective-redundancy with modern threshold cryptographic techniques, to form an impregnable backdoor containment unit. To evaluate our design, we implement a prototype that provides the highest level of assurance possible with COTS components. Specifically, we employ more than a hundred COTS processing units, verified to FIPS140-2 Level 4 tamper-resistance standards, and use them to realize high-confidentiality random number generation, key derivation, public key decryption and signing. Our experiments show a reasonable computation overhead (e.g., less than 0.8% for Decryption and 17% for Signing), and an almost-linear trade-off between throughput and backdoor-tolerance.

4.1 Introduction

Integrated Circuits (ICs) are ubiquitous and used in critical applications ranging from automotives, communication networks to military equipment and space stations. For these systems to be secure and reliable, the design and fabrication of the underlying hardware and software must abide to high-quality specifications and standards. These ensure that the software has no intentional and unintentional bugs, but more importantly ensure the integrity of the hardware supply chain: errors in the ICs would be devastating for the security of the final system [48].

Unfortunately, IC manufacturers are not always able to oversee all parts of the supply chain [34, 55]. The constant reduction of transistor size that makes IC fabrication a very expensive process, and forces IC designers to outsource the fabrication part to overseas foundries (for cost reduction) [30, 43, 88]. As a result, a part of the ICs' supply chain is left vulnerable to attacks from malicious insiders [75, 60, 57, 12] and has a higher probability of introducing unintentional errors in the final product. There are several documented real-world cases where hardware contained undocumented functionality that classified as a backdoor or a trojan horse. In many cases, the affected ICs are used in high-confidentiality and integrity applications, such as military [68, 54], networking

equipment [46, 36], and various others [67, 2, 53, 66]. Moreover, there are various types of hardware trojans (HT), and backdoors described by the academic community that demonstrate the extent of the problem and mitigation difficulty [81, 80, 82, 50, 15, 59, 11, 19]. Understandably, one of the classes of systems that can be particularly disrupted by such errors is those relying on cryptographic hardware. Such systems (e.g., Hardware Security Modules and Cryptographic Accelerators) are widely used in real-world deployments with high security needs (e.g., banking applications), where they carry out sensitive security tasks (e.g., key generation and storage, document signing) and provide a protection layer against cyber-attacks and security breaches. Furthemore, this work is also motivated by the need for high-assurance mix network infrastructures [29] that require secure storage of keys and high-throughput key derivation operations despite extremely powerful adversaries.

Due to the severity of these threats, there is a large body of work on the mitigation of malicious circuitry. Existing research branches into two different directions: detection and prevention. Detection techniques aim to determine whether any HTs exist in a given circuit [86, 84, 69, 3], while prevention techniques either impede the introduction of HTs, or enhance the efficiency of HT detection [78, 24, 83, 62, 77]. Unfortunately, both detection and prevention techniques are brittle, as new threats are able to circumvent them quickly [83]. For instance, analog malicious hardware [87] was able evade known defenses, including split manufacturing, which is considered one of the most promising and effective prevention approaches. Furthermore, most prevention techniques come with a high manufacturing cost for higher levels of security [16, 83, 24], which contradicts the motives of fabrication outsourcing. To make matters worse, vendors that use commercial off-the-shelf (COTS) components are constrained to use only post-fabrication detection techniques, which further limits their mitigation toolchest. All in all, backdoors being triggered by covert means and mitigation countermeasures are in an arms race that seems unfavorable to the defenders [85, 83].

It has been repeatedly noted that computing monoculture increases the exposure of infrastructure to attacks [70, 37, 92]. One solution to this problem is offered by N-variant systems. Such systems achieve heterogeneity by including multiple randomized system copies within the same design and use them in parallel to replicate all computations. Their insight is that non-identical copies prevent exploits from adjusting themselves to attack all of them simultaneously. However, existing works only consider external attacks and do not account for adversaries residing in the supply chain, or introducing malicious circuitry to the component itself.

In this paper, we build high-assurance hardware from a set of untrusted components, as long as at least one of them is not compromised, even if benign and malicious components cannot be distinguished.

Our key insight is that by combining established privacy enhancing technologies (PETs) with mature fault-tolerant system architectures, we can distribute trust between multiple components originating from non-crossing supply chains, thus reducing the likelihood of compromises.

To achieve that, we deploy distributed cryptographic schemes on top of an N-Variant system architecture, and build a trusted platform that supports a wide-range of commonly used cryptographic operations (e.g., random number & key generation, decryption, signing).

This design draws from protective-redundancy and component diversification [23] and is built on the assumption that multiple processing units and communication controllers may be compromised by the same adversary. However, unlike N-Variant systems, instead of replicating the computations on all processing units, Myst uses the homomorphic properties of our cryptographic schemes to distribute the secrets so that the components hold only shares of the secrets (and not the secrets themselves), at all times. This entails that as long as one of the components holding shares remains honest the secret cannot be reconstructed or leaked. We also note that existing detection and prevention techniques can work complimentary to our proposed design as a mean to reduce the

likelihood of compromises for individual components.

To our knowledge this is the first work to use distributed cryptographic protocols to build hardware that is tolerant to multiple components carrying trojans or errors. To summarize, this paper makes the following contributions:

- **Concept:** We introduce *backdoor-tolerance*, where a system can still preserve its security properties in the presence of multiple compromised components.
- **Design:** We demonstrate how cryptographic schemes (§4.4) can be combined with N-Variant system architectures (§4.3), to build high-assurance systems.
- **Implementation:** We implement the proposed architecture by building a custom board featuring 120 highly-tamper resistant ICs, and implement secure variants of random number & key generation, public key decryption and signing (§4.5).
- **Optimizations:** We implement a number of optimizations to ensure the Myst architecture is competitive in terms of performance compared to single ICs. Some optimizations concern embedded mathematical libraries which are of independent interest.
- **Evaluation:** We quantitatively evaluate the performance of Myst, and use micro-benchmarks to assess the cost of all operations and bottlenecks (§4.6).

The rest of the paper is organized as follows. Section 4.2 defines *backdoor-tolerance*, our threat model and assumptions. The Myst architecture is introduced in Section 4.3, and our distributed cryptosystem is detailed in Section 4.4. Section 4.5 describes the implementation of our prototype, while Section 4.6 measures its performance. Finally, related works and their relation to Myst are discussed in Section 4.7, while Section 4.8 concludes the paper.

4.2 Preliminaties

In this section, we introduce *backdoor-tolerance*, and outline our security assumptions for adversaries residing in the IC's supply chain.

4.2.1 Definition

A *Backdoor-Tolerant* system is be able to ensure the confidentiality of the secrets it stores and the integrity of the computations it performs. Such a system can prevent data leakages, protect the integrity of the generated keys and random numbers, and identify misbehaving components. Note that the definition makes no distinction between honest design and fabrication mistakes and hardware trojans, and accounts only for the impact of these errors on the security of the system.

4.2.2 Threat model

We assume an adversary is able to incorporate errors in some ICs. Such an adversary can gain access to the secrets stored in the affected ICs and may also breach the integrity of any cryptographic function run by the IC (e.g., random number generation). However, we also assume that not all ICs are compromised either due to the use of different fabrication locations, or different vendors.

Moreover, the adversary has full control over the communication buses used to transfer data to and from the ICs. Hence, the adversary is able to exfiltrate any information on the bus and the channel controller, and inject and drop messages to launch her attacks. Additionally, she is able to connect and issues commands to the ICs, and if a system features more that one malicious ICs, she

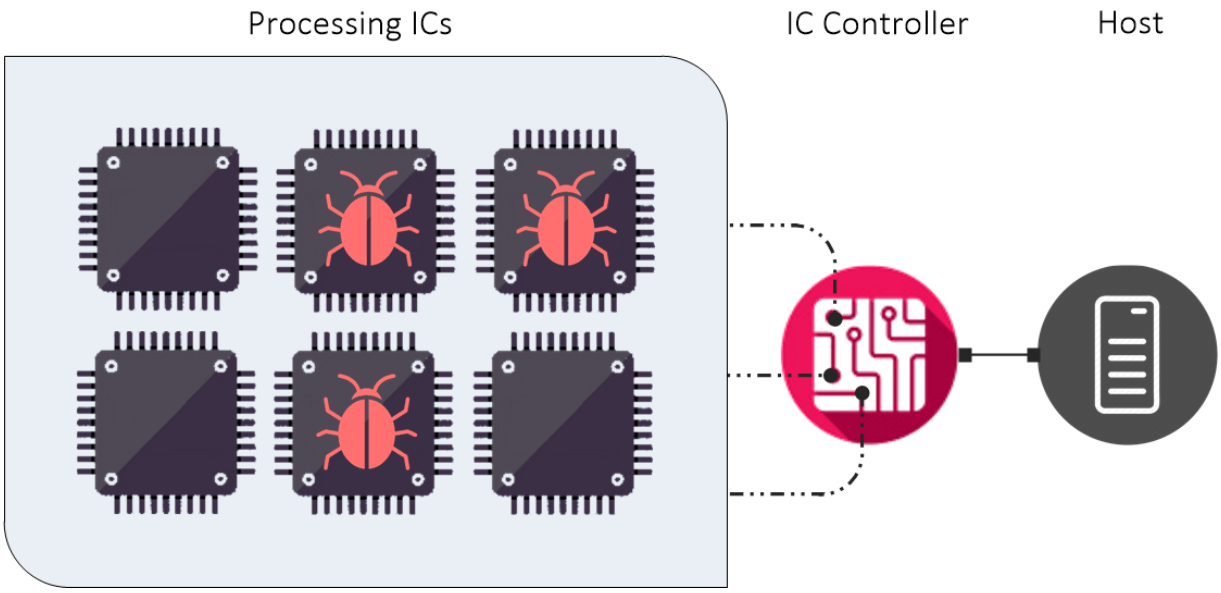

Figure 4.1: An overview of the Myst's distributed architecture, featuring all the integral components and communication buses.

is able to orchestrate attacks using all of them. We also assume that the adversary may use any other side-channel that a Trojan has been programmed to emit on – such as RF [5], acoustic [38] or optical [6] channels. However, we assume that side-channels of chips not containing backdoors may not leak any useful information to the adversary.

We make standard cryptographic assumptions about the adversary being computationally bound, and not being able to solve through any shortcuts standards cryptographic problems, such as the Decisional Diffie-Hellman problem [18], etc.

Attacks aiming to simply compromise the availability of the system, are out of the scope of the paper. Such attacks can be easily thwarted through standard replication of the component, without any complications for confidentiality. Instead, we assume that the malicious ICs aims to be as stealthy as possible, and conceal their presence as best as possible. Moreover, we also assume that malicious components are indistinguishable from benign ones.

Finally, we assume a software developer builds and signs the applications to be run on the ICs. From our point of view they are trusted to provide high-integrity software without backdoors or other flaws. This integrity concern may be tackled though standard high- integrity software development techniques, such as security audits, public code repository trees, deterministic builds, etc.

4.3 Architecture

In this section, we introduce the Myst architecture, which draws from fault-tolerant designs and N-variant systems [72, 23]. The proposed design is based on the thesis that given a diverse set of k ICs sourced from k semi-trusted suppliers, a trusted system can be build, as long as at least one of them does not contain intentional backdoors or unintentional errors. As illustrated in Figure 4.1 Myst has three types of components: (1) a remote host, (2) an untrusted IC controller, and (3) the processing ICs.

Processing ICs. The ICs form the core of Myst, as they are collectively entrusted to perform high-integrity computations, and provide secure storage space. Each implementation must feature two or more ICs, of which at least one must be free of backdoors. We define this coalition of ICs, as a quorum. The exact number of ICs in a quorum is determined by the user depending on the degree of assurance she wants to achieve (see also the Assessment part of Section 4.6). The ICs are programmable processors or microprocessors, have enough persistent memory to store keys and feature a secure random number generator. Protection against physical tampering or side-channels is in most cases also highly desirable and mandated by the industry standards and best practices for cryptographic hardware. For this reason, our prototype (Section 4.5), comprises of components that verify to the highest level of tamper-resistance (i.e., FIPS140-2 Level 4), and feature a very reliable random number generator (i.e., FIPS140-2 Level 3).

IC controller. The controller enables the communication between the ICs themselves, and makes Myst accessible to the outside world. It manages the bus that the processing ICs use to communicate with each other and the interface where incoming requests from the user are delivered. Specifically, it enables:

- *Unicast, IC-to-IC*: the ICs are able to send instructions to each other, and receive the outcome.
- *Broadcast, IC-to-ICs*: the ICs are able to broadcast instructions to all other ICs, and receive the outcomes.
- *Unicast, Host-to-IC*: remote clients are able to send commands to specific ICs, and receive the outcome.
- *Broadcast, IC-to-ICs*: remote clients are able to broadcast commands to all ICs, and receive the outcomes.

The controller is also an IC, and other hardware, and it may also feature intentional or unintentional errors or backdoors. For instance it may drop, modify packets or forge new ones, in order to launch attacks against the protocols executed. It may also collude with one or more processing ICs.

Remote host. The remote host connects to Myst through the IC controller; it issues high level commands and retrieves the results. The remote host can be any kind of computer either in the local network or in a remote one. In order for the commands to be executed by the ICs, the host must provide proof of its authorization to issue commands, usually by signing them with a public key associated with the user's identity (e.g., Certificate by a trusted CA). Each commands issued must include: 1) the requested operation, 2) any input data, and 3) the host's signature (see also Section 4.3.1). We note that, the remote host treats Myst as a black box and simply submits service requests, without assuming any knowledge of its internal workings. Instead, the details regarding the formation of quorums are determined by the system operator or the owner of the secrets.

Communication channels. At the physical level, the ICs, the controller and the hosts are connected through buses and network interfaces. These are visible to the adversary, but we use established cryptographic mechanisms to ensure integrity and confidentiality for transmitted data.

More specifically, all unicast and broadcast packets are signed using the sender IC's certificate, so that their origin and integrity can be verified by recipients. Moreover, in cases where the confidentiality of the transmitted data needs to be protected, encrypted end-to-end channels are also established.

4.3.1 Access control layer

Access Control (AC) is critical for all systems operating on confidential data. In Myst, AC determines which hosts can submit service requests to the system, and which quorums they can interact with. Despite the distributed architecture of Myst, we can simply replicate established AC techniques on each IC. More specifically, each IC is provided with the public keys of the hosts that are allowed to have access to its shares and submit requests. Optionally this list may distinguish between hosts that have full access to the system, and hosts that are allowed to perform only a subset of the operations. Once a request is received the IC verifies the signature of the host against this public key list. The list can be provided either when setting up Myst, or when storing a secret or generating a key.

We note that it is up to the entity handling the keys to decide the parameters of each quorum (i.e., size k, ICs participating), and provide the AC lists to the ICs. This is a very critical procedure, as if one of the hosts turns out to be compromised, the quorum can be misused in order to either decrypt confidential ciphertexts or sign bogus statements. However, in any case the secrets stored in the quorum will remain safe as there is no way for the adversary to extract them, unless they use physical-tampering, which our prototype (§4.5) is also resilient against).

4.4 Secure distributed computations

In this section, we introduce a set of protocols that leverage the diversity of ICs in Myst to realize standard cryptographic operations manipulating sensitive keying material. In general, we could use techniques from secure multi-party computation (SMPC) [28, 27] to distribute secret computations across all ICs and ensure they remain confidential even in the face of some compromised components. However, we instead leverage higher-level threshold cryptographic protocols, that offer directly important cryptographic functionality and are secure under our threat model. More specifically, the cryptosystem comprises of a key generation operation (§4.4.1), the ElGamal encryption operation (§4.4.2), distributed ElGamal decryption (§4.4.3), and signing based on Schnorr signatures (§4.4.4) [31, 33]. These operations are realized using interactive cryptographic protocols that rely on standard cryptographic assumptions such as the hardness of the discrete logarithm problem in elliptic curve groups.

Prior to the execution of any protocols, the ICs must be initialized with the domain parameters $T = (p, a, b, G, n, h)$ of the elliptic curve to be used, where p is prime specifying the finite field $\mathbb{F}_p$, a and b are the curve coefficients, G is the base point of the curve, with order n, and h is the curve's cofactor. More details on the technical aspects of the initialization procedure are provided in the provisioning section 4.6.3.

Trade-off between confidentiality & availability. All our distributed protocols use a threshold k, which determines how many key shares must be combined in order to reconstruct a distributed key. In practice, k is defined by the user and expresses the trade-off between confidentiality and robustness: When the k is equal to the number of processing ICs t, then secrets are safe in the presence of $t - 1$ compromised ICs. On the other hand, a lower $k < t$ enables the system to remain fully functional even if some of the ICs fail – maliciously or through wear. Hence, the security guarantees offered by our distributed protocols is determined by the threshold k. In this work, we chose to maximize confidentiality, and resist the presence of $t - 1$ actively malicious ICs – which also keeps the presentation of our protocols simpler. However, in cases where reliability is also important (e.g., to withstand IC failures) the security level can be adjusted in favor of redundancy by using appropriate threshold schemes [58].

Distributed key pair generation. The Distributed Key Pair Generation (DKPG), is a key building block for most other protocols. In a nutshell, DKPG enables a *quorum* Q of t ICs to collectively generate a random secret x, which is an element of a finite field and a public value $y = x \cdot G$ for a given public elliptic curve point G. At the end of the DKPG protocol, each player holds a share of the secret x, denoted as x_i and the public value y. All steps of the protocol are illustrated in Figure 4.2, and explained in turn below.

The execution of the protocol is triggered when an authorized host sends the corresponding instruction (❶). At the first step of the protocol, each member of Q runs Algorithm 1 and generates a triplet consisting of: 1) a share x_i, which is a randomly chosen element of $\mathbb{Z}_n$, 2) an elliptic curve point y_i, and 3) a commitment to y_i denoted h_i. (❷)

Algorithm 4.4.1: TripletGen: Generation of a pair and its commitment.

Input : The domain parameters λ
Output: A key triplet (x_i, y_i, h_i)
1 $x \leftarrow \text{Rand}(\lambda)$
2 $y \leftarrow x_i \cdot G$
3 $h \leftarrow \text{Hash}(y_i)$
4 **return** (x_i, y_i, h_i)

Upon the generation of the triplet, the members perform a pairwise exchange of their commitments (❸), by the end of which, they all hold a set $H = \{h_1, h_2, .., h_t\}$. The commitment exchange terminates when $|H_q| = t \ \forall q \in Q$, and another round of exchanges starts, this time, for the shares of y (❹) $Y = \{y_1, y_2, .., y_t\}$. The commitment exchange round is of uttermost importance as it forces the participants to commit to a share of y, before receiving the shares of others. This prevents attacks where an adversary first collects the shares of others, and then crafts her share so as to bias the final pair, towards a secret key they know.

Algorithm 4.4.2: CommitVerify: Checks y_i, against their respective commitments.

Input : (Y, H)
Output: Bool
1 **for** $i \in \{1, \ |Y|\}$ **do**
2 | **if** $Hash(y_i) \neq h_i$ **then**
3 | | **return** False
4 | **return** True

Once $|Y_q| = t \ \forall q \in Q$, the member executes Algorithm 2 to verify the validity of Y's elements against their commitments in H. (❺) If one or more commitments fail the verification then the member infers that an error (either intentional or unintentional) occurred and the protocol is terminated. Otherwise, if all commitments are successfully verified, then the member executes Algorithm 3 (❻) and returns the result to the remote host (❼). Note that, it is important to return y, as well as the individual shares y_i, as this protects against integrity attacks, where malicious ICs return a different share than the one they committed to during the protocol [58, 39]. Moreover, since y_i are shares of the public key, they are also assumed to be public, and available to any untrusted party.

In the following sections, we rely on DKPG as a building block of more complex operations, such as the key generation operation (Section 4.4.1) and the distributed signing operation (Section 4.4.4).

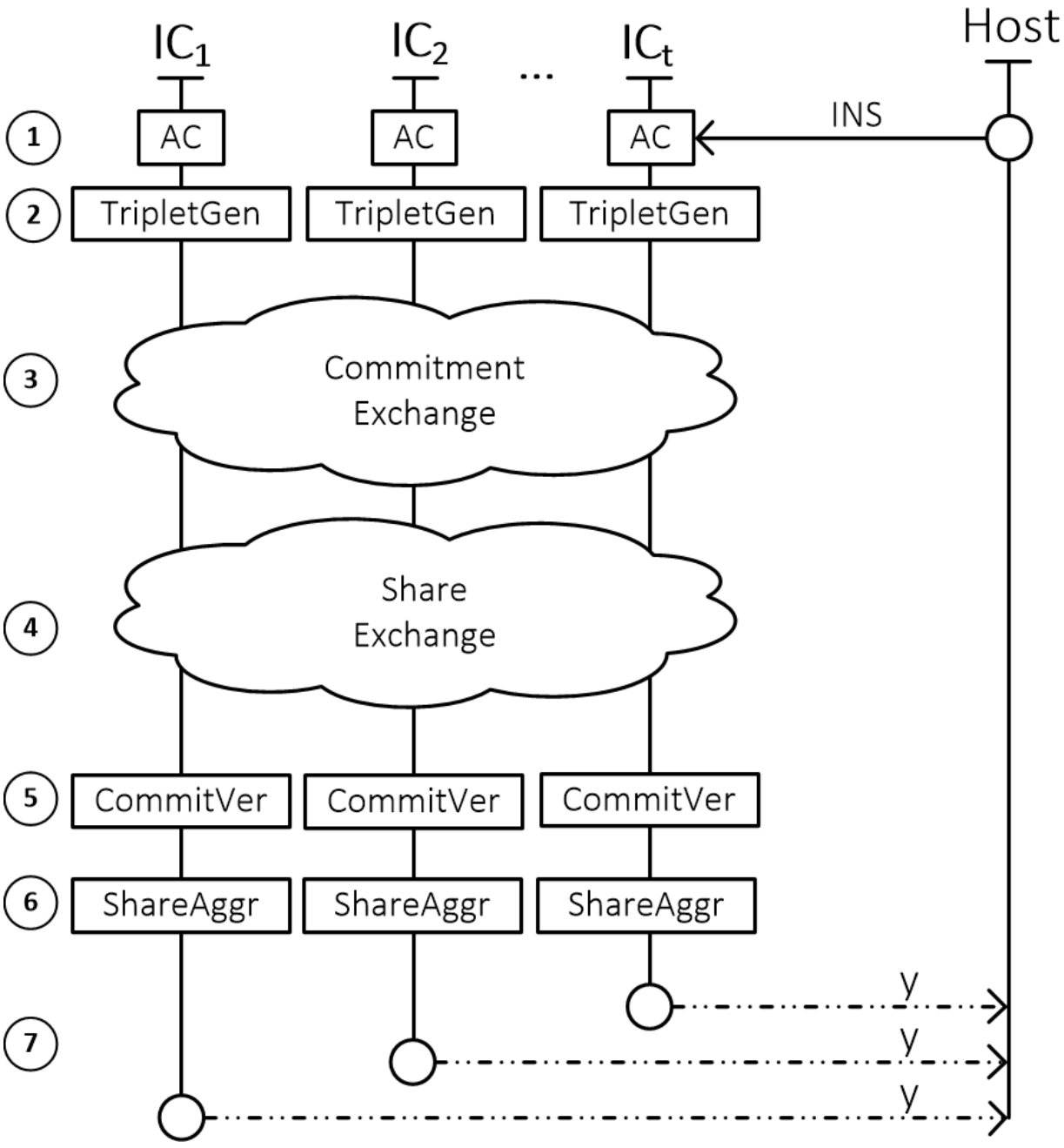

Figure 4.2: The interaction between the different participants during the execution of the Distributed Key Pair Generation (DKPG) protocol.

Algorithm 4.4.3: ShareAggr: Aggregates public keys to form a common key.

Input : (Y)
Output: The aggregate of the public keys y

$y \leftarrow \varnothing$
for $y_i \in Y$ **do**
  $y \leftarrow y + y_i$
return y

4.4.1 Distributed public key generation

The distributed key generation operation enables multiple ICs to collectively generate public and private key shares that are only known to each IC. This is key in the presence of hardware trojans as single ICs do not have access to the full private key at any point, and the integrity and secrecy of the pair remains immune to malicious share contributions.

We opt for a scheme that offers the maximum level of confidentiality (*t-of-t*), and whose execution is identical to DKPG seen in Figure 4.2. However, there are numerous protocols that allow for different thresholds, such as Pedersen's VSS scheme [58, 39, 71].

Once a key pair has been generated, the remote host can encrypt a plaintext using the public key y, request the decryption of a ciphertext, or ask for a plaintext to be signed. In the following sections we will outline the protocols that realize these operations.

4.4.2 Encryption

For encryption we use the Elliptic Curve ElGamal scheme [31, 20] (Algorithm 4). This operation does not use the secret key, and can be performed directly on the host, or remotely by any party holding the public key, hence there is no need to perform it in a distributed manner.

Algorithm 4.4.4: Encrypts a plaintext using the agreed common public key.

Input : The plaintext m encoded as an element of the group$\mathbb{G}$, and the calculated public key Y_{agg}
Output: The Elgamal ciphertext tuple, $(C_1,\ C_2)$

$r \leftarrow Rand(T)$
$C_1 \leftarrow r \cdot G$
$C_2 \leftarrow m + r \cdot Y_{agg}$
return $(C_1,\ C_2)$

4.4.3 Decryption

One of the fundamental cryptographic operations involving a private key is ciphertext decryption. In settings, where the key is held by a single authority, the decryption process is straightforward, but assumes that the hardware used to perform the decryption does not leak the secret decryption key. The Myst platform addresses this problem by distributing the decryption process between k distinct processing ICs that hold shares of the key (Figure 4.3).

The protocol runs as follows: Initially, the host broadcasts the decryption instruction along with the ciphertext tuple $\langle C_1,\ C_2 \rangle$ to the processing ICs (❶). Upon reception, the ICs first verify that the request is signed by an authorized user (❷), and then execute Algorithm 5 to generate their

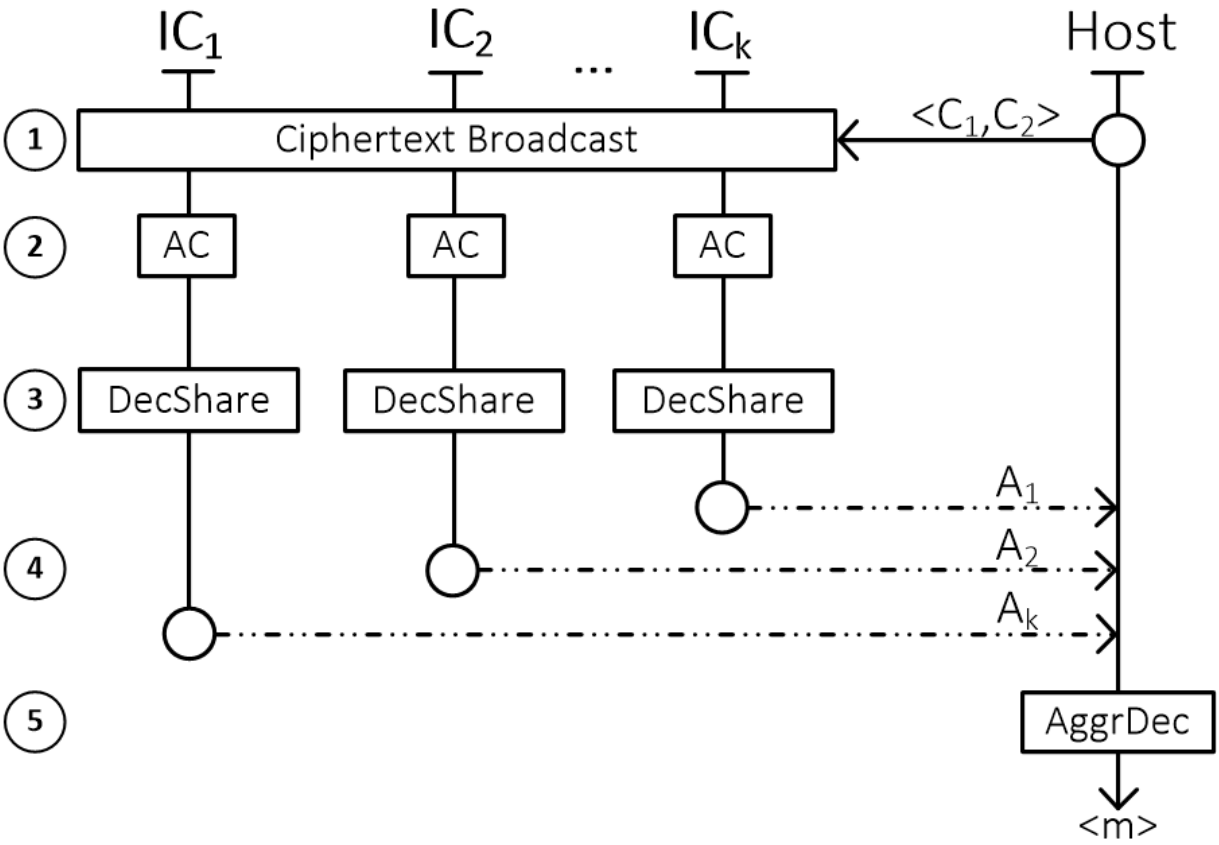

Figure 4.3: The interaction between the different ICs during the execution of the distributed decryption protocol.

decryption shares A_i (❸). Once the shares are generated they are sent back to the host, signed by the ICs and encrypted under the host's public key (❹). Once the host receives k decryption shares, it executes Algorithm 6 to combine them and retrieve the plaintext (❺).

Algorithm 4.4.5: DecShare: Returns the decryption share for a given ciphertext.

Input : The Elgamal ciphertext (C_1, C_2) and the IC's private key of x_i.
Output: The decryption share A_i, where i is the IC's uid

$A_i \leftarrow -x_i \cdot C_1$
return A_i

Algorithm 4.4.6: AggrDec:Combines the decryption shares and returns the plaintext for a given ciphertext.

Input : The Elgamal ciphertext C_2 and the set of decryption shares A.
Output: The plaintext m

$D \leftarrow \varnothing$
for $A_i \in A$ **do**
  $D \leftarrow D + A_i$
$m \leftarrow (C_2 + D)$
return m

It should be noted that during the decryption process, the plaintext is not revealed to any other party except the host, and neither the secret key nor its shares ever leave the honest ICs. An extension to the above protocol can also prevent malicious ICs from returning arbitrary decryption shares, by incorporating a non-interactive zero knowledge proof [17] in the operation output.

4.4.4 Signing

Apart from key generation, encryption and decryption, Myst also supports distributed signing – an operation that potentially manipulates a long term signature key.

Algorithm 4.4.7: SigShare: Returns the signature share for a given plaintext.

Input : The Elliptic Curve parameters λ, the plaintext to be signed m, the IC's private key of x_i, the IC's randomness share r_i, and the aggregated random EC point R.
Output: The signature share tuple (s_i, e)

$e \leftarrow \text{Hash}(m||R)$
$s_i \leftarrow r_i - x_i \cdot e \mod n$
return (s_i, e)

Initially, all k ICs cooperate to generate a public key y using the distributed key generation operation (Section 4.4.1), and store securely their own key share x_i. From this point on, for each message m, which is to be signed (❶), the host's authorization is verified (❷), and the following protocol is executed: At first, a fresh random commitment R is generated in a distributed manner, using the DKPG protocol and each IC holds the commitment R and a share of the witness r_i, where $R = \sum_{i=0}^{t}(r_i \cdot G)$ (❸). Then each member of the quorum computes and broadcasts a share of m's signature $\sigma_i = \langle s_i, e \rangle$, using Algorithm 7(❹). At this point, the remote host can combine these k shares and compute the signature σ. More specifically, given a vector of signature shares (❺), the

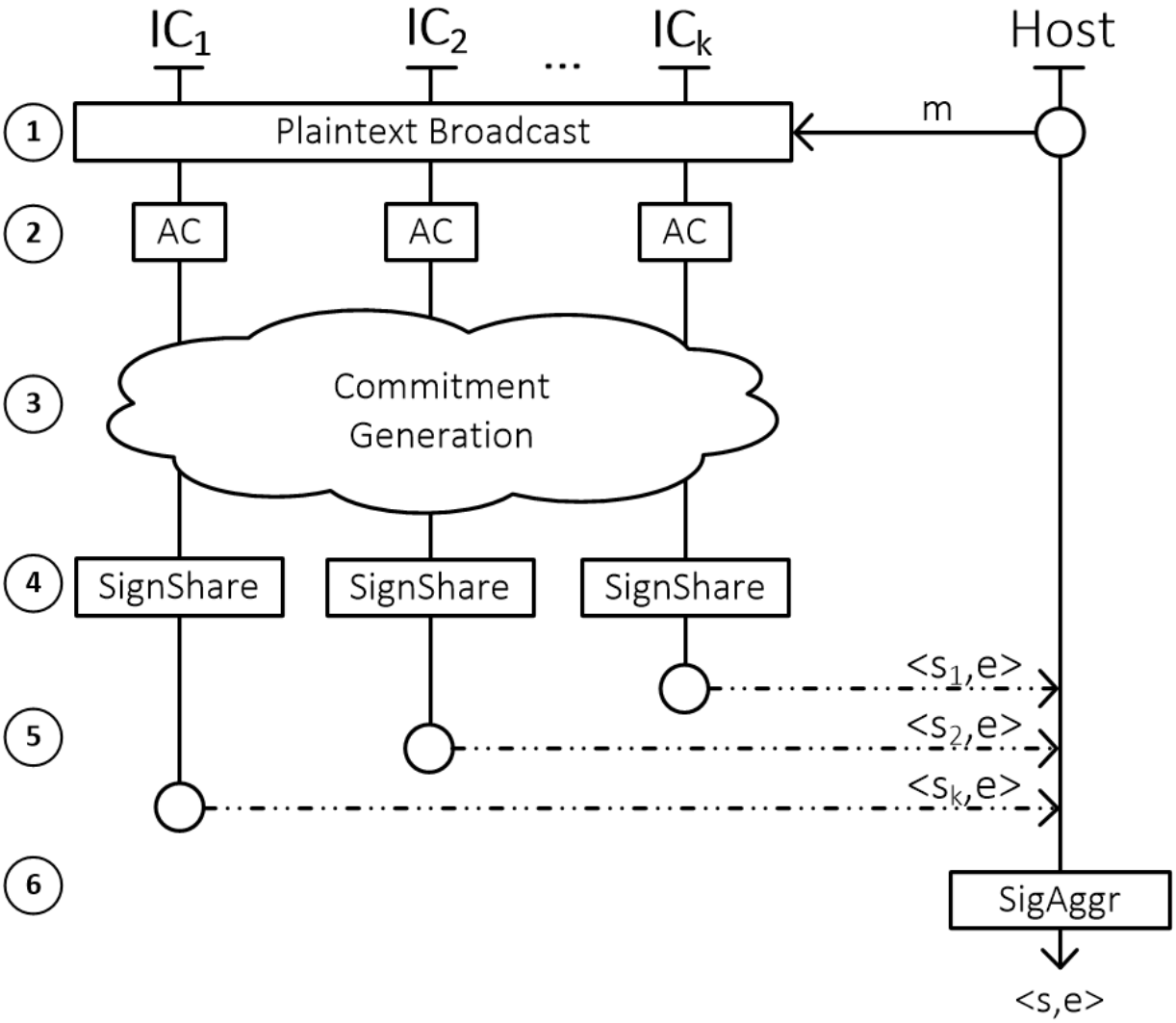

Figure 4.4: The interaction between the different players during the execution of the distributed signing protocol.

signature $\sigma = \langle s, e \rangle$, is computed through $s = \sum s_i \mod n \; \forall s_i \in \vec{\sigma}$ (❻). The recipient of $\langle m, \sigma \rangle$ can verify the validity of the signature by checking if $e = \text{Hash}(m||r_v)$, where $r_v = s \cdot G + e \cdot Y$

4.4.5 Random number generation

Another important application of secure hardware is the generation of a random fixed-length bit-strings in the presence of active adversaries. The key property of such systems is that errors (e.g., a hardware backdoor), should not allow an adversary to bias the generated bitstring.

The implementation of such an operation is straightforward. The remote host submits a request for randomness to all actors participating in the quorum. Subsequently, each actor independently generates a random share b_i, encrypts it with the public key of the host, and signs the ciphertext with its private key. Once the host receives all the shares, he combines them to retrieve the b and then uses an one way function (e.g., SHA3-512 [14]) to convert it to a fixed length string.

4.5 Implementation

In this section, we provide the implementation details of our Myst prototype. We first focus on the custom hardware we built, and outline its different components and capabilities. Thereafter, we discuss the development of the software for the semi-trusted ICs and the details of the remote host.

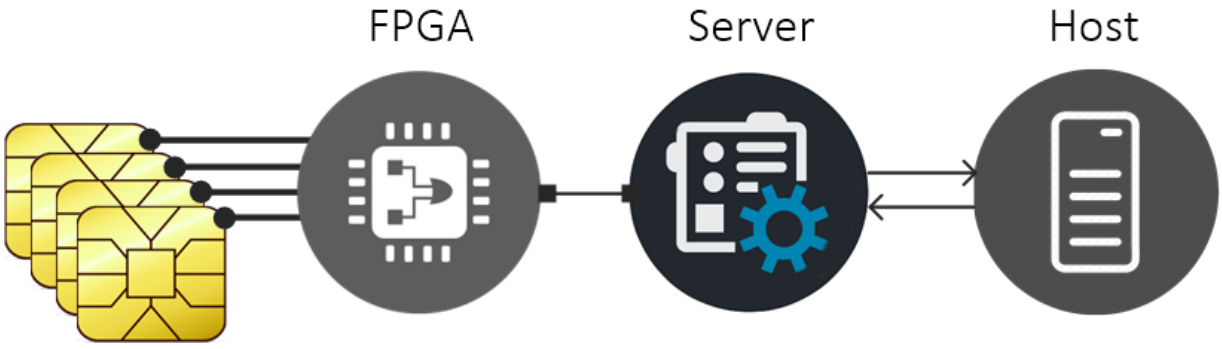

Figure 4.5: Overview of Myst's components.

4.5.1 Hardware design & implementation

For our Myst prototype, we designed our own custom circuit board, which features 120 processing ICs interconnected with dedicated buses with 1.2Mbps bandwidth.

The processing ICs are JavaCards (version 3.0.1), loaded with a custom applet implementing the protocols outlined in Section 4.4. JavaCards are an ideal choice as they provide excellent interoperability (i.e., applets are manufacturer-independent), which contributes to IC-diversification and prevents lock-in to particular vendors. Moreover, they also fulfill all the requirements listed in Section 4.2: 1) they are tamper-resistant (FIPS140-2 Level 4, CC EAL4) and can withstand attacks from adversaries with physical access to them [63], 2) they feature secure (FIPS140-2 compliant) on-card random number generators and 3) they offer cryptographic capabilities (e.g., Elliptic curve addition, multiplication) through a dedicated co-processor. In addition to these, they have secure and persistent on-card storage space, ideal for storing key shares and protocol parameters.

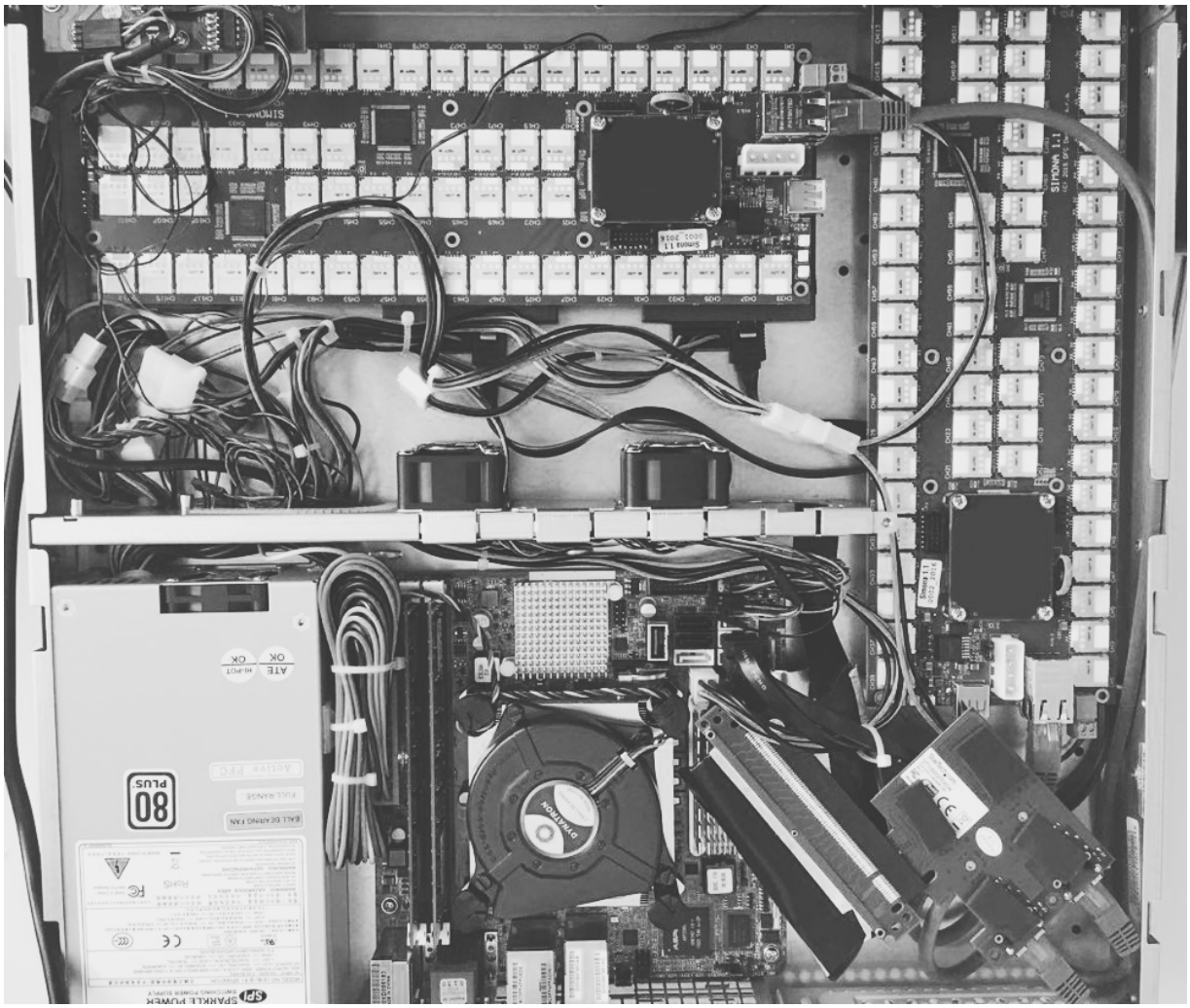

Figure 4.6: Myst's prototype can support more than 120 ICs, and fits in a single rack unit casing.

The host is implemented using a computer that runs a python client application, which submits the user requests (e.g., Ciphertext Decryption) to Myst using a RESTful API exposed by the device. The incoming requests are handled by a Spring server, which parses them, converts them to a sequence of low-level instructions, and then forwards these to the IC controller, through an 1Gbps TCP/UDP interface.‘ The ICs controller, a programmable Artix-7 FPGA, listens for the incoming instructions and then routes them to the processing IC, through separate physical connections. We took special care that these buses offer a high bandwidth (over 400kbps), to prevent bottlenecks even in extreme use cases. Once the ICs return the results, the controller communicates them back to the server, that subsequently forwards them to the host.

4.5.2 Software

We implement the protocols of Section 4.4 and provide the necessary methods for inter-component communication and system parameterization.

We develop and load the processing ICs with JavaCard applets implementing methods for: 1)

Card Management, 2) Key Generation, 3) Decryption, and 4) Signing. Although JavaCard APIs since version 2.2.2 specifies a BigNumber class, this class is either unsupported by real cards or provides only too small internal type length (typically at most 8 bytes). The only third-party BigInteger library available (i.e., BigNat[1]) is unmaintained and lacked essential operations. Moreover, low-level elliptic curve operations are poorly supported as well. This made the implementation of our cryptographic schemes tedious. For this reason, we extend BigNat to provide methods catering to our specific needs and develop our own EC operations library. Our EC library provides methods for EC point initialization, negation, addition, subtraction and scalar multiplication.

For EC point addition we use an NXO API, while scalar multiplication is achieved through a trick: we use the high-level Key Agreement EC Diffie-Hellman method, exposed by the standard JavaCard API, in order to perform scalar EC multiplication on the cryptographic co-processor, instead of the slower general-purpose processor of the card. Our current implementation is based on the NIST P-256 [76, 1] curve that provides at least 128 bits of security. However, it can also be trivially adapted for any other curve.

Optimizations. We optimize our JavaCard applet for speed and prevent side-channel attacks. Although JavaCard applets are compiled with standard Java compiler, common Java implementation patterns (e.g., frequent array reallocation due to resizing) are prohibitively expensive on JavaCards. Therefore, we use the following optimization techniques based on good practices and performance measurements from real, non-simulated, smart cards [73]:

- ❖ We use hardware accelerated cryptographic methods from JavaCard API instead of custom implementations interpreted by JavaCard virtual machine.
- ❖ We store the data in the faster RAM-allocated arrays instead of persistent, but slower EEPROM/Flash
- ❖ We use copy-free methods which minimize the move of data in memory, and also utilize native methods provided by JCSystem class for array manipulation like memory set, copy and erase.
- ❖ We made sure to pre-allocate all cryptographic engines and key objects during the applet installation. No further allocation during the rest of the applet lifetime is performed.
- ❖ Surplus cryptographic engines and key objects are used to minimize the number of key scheduling and initialization of engine with a particular key as these operations impose non-trivial overhead [73].
- ❖ We refrain from using single long-running commands which would cause other cards to wait for completion, thus increasing the latency of the final operation.

Finally, we optimized two fundamental operations commonly used in cryptographic protocols: 1) integer multiplication, and 2) the modulo operation optimized for 32 byte-long EC coordinates. This was necessary, as the straightforward implementation of these two algorithms in JavaCard bytecode is both slow and potentially vulnerable to side-channel timing attacks. We, instead, implemented both operations so as to use the native RSA engine and thus have constant-time run-time.

The integer multiplication of a and b can be rewritten as $a \cdot b = ((a+b)^2 - a^2 - b^2)/2$. The squaring operation (e.g., a^2) can be quickly computed using a pre-prepared raw RSA engine with a public exponent equal to 2 and a modulus n, that is larger than the sum of the lengths of both operands. On the other hand, the integer modulo of two operands a (64 bytes long) and b (32 bytes long) is not so straighforward. We exploit the fact that b is always the order of the fixed point G

[1] `https://ovchip.cs.ru.nl/OV-chip_2.0`

in the P-256 Elliptic Curve [76, 1], and transform $a \mod b = a - (((a \cdot x) \gg z) \cdot x)$ where x and z values are pre-computed offline [41]. As a result, a modulo operation can be transformed to two RSA operations, one shift (with z being multiple of 8) and one subtraction. Note that we cannot directly use RSA with a public exponent equal to 1 as operands are of different length and also shorter than smallest RSA length available on the card.

4.5.3 System states

Initially, the system is in an non-operational state, where the processing ICs do not respond to user requests. To make it operational, a secure initialization process has to be carried out. During the initialization the processing ICs and the other components described in 4.3 are loaded with verified application packages, and the domain parameters for the distributed protocols are set. Moreover, the ICs are provided with their certificates that they will later use to sign their packets and establish secure communication channels.

Once the initialization process has been completed, the system switches to an operational state and is available to serve user requests. Depending on the configuration, the system may be brought back to an uninitialized state, in order to load new software or change the protocol parameters. We further discuss, the importance of the initialization process in Section 4.6.3.

4.6 Evaluation

In this section, we evaluate our Myst prototype by studying its performance in two aspects:

- **Performance.** We compare the latency and throughput of our prototype to that of a single-IC system (§4.6.1).
- **Scalability & extensibility.** We determine the effectiveness of distributing incoming operation requests between multiple quorums and enabling simultaneous card-to-card communication (§4.6.2)

Experimental setup. All evaluations were performed using the setup illustrated in Figure 4.5. The host is a single machine with a CentOS 7 OS (3.10.0 Linux kernel), featuring a 3.00GHz Intel(R) Core i5-4590S CPU, 16GB of RAM, and uses a python client application to submit service requests to Myst, through a 1Gbps Ethernet interface (as described in Section 4.5). Upon reception, the server uses the Spring framework [47] to parse them, and then forward the instructions to the Artix-7 FPGA, which subsequently routes them to the individual smart cards.

In all experiments, we collect response-time measurements from both the host and the Spring server. On average the round-trip from the host to the server is $5ms$. This is due to the high-bandwidth interface connecting the two machines. For accuracy, we use the host measurements in the rest of the section.

4.6.1 Performance impact

In this subsection we evaluate the performance impact of Myst, and compare its throughput and latency with that of a single-IC system. Moreover, we examine the impact of our optimizations on the overall performance of the system.

Methodology. To better understand the overhead that the use of a distributed architecture entails we run experiments that measure the latency, as the size of the protocol quorum grows. We first submit 1,000 requests for each cryptosystem operation (Section 4.4) in one JavaCard and

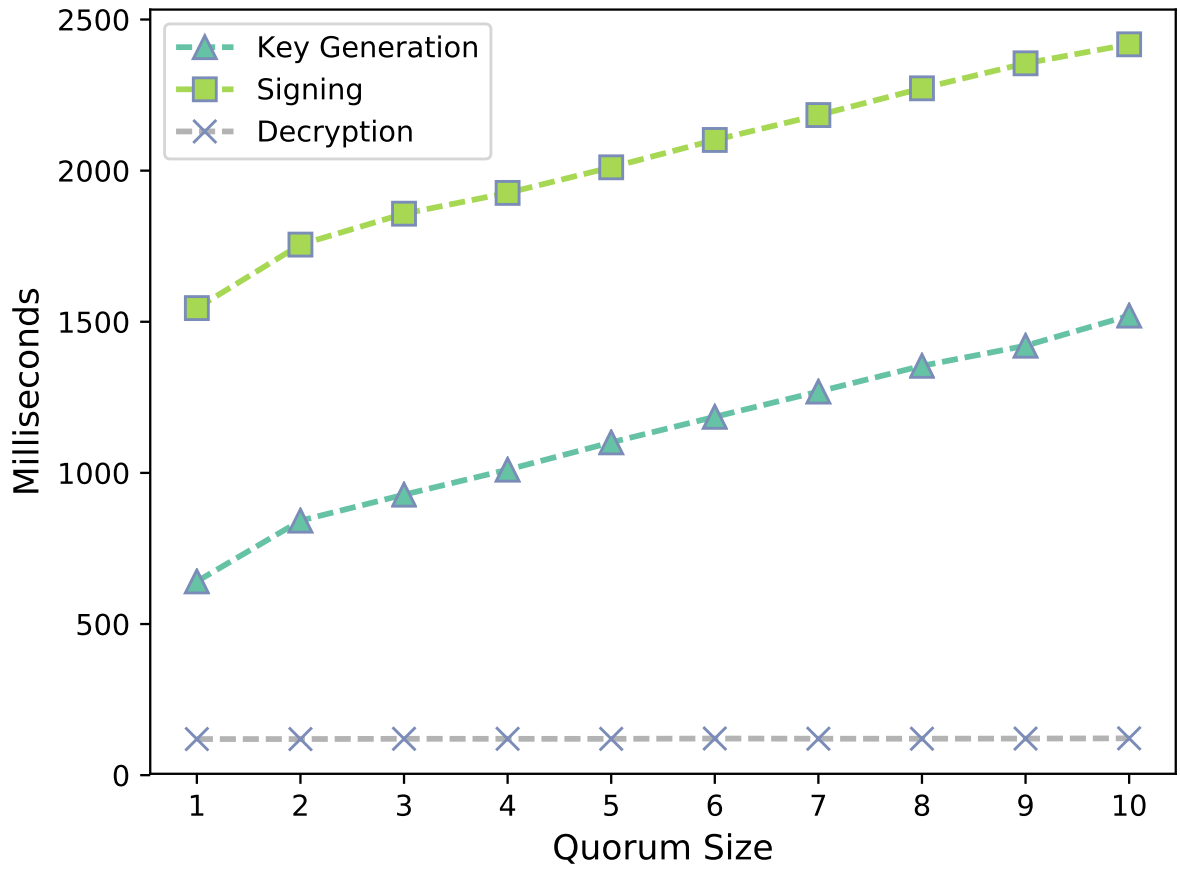

Figure 4.7: The average response time for each distributed operation of the cryptosystem, in relation to the quorum (i.e., a coalition of multiple ICs) size.

measure the response time. We then extend the experiment to larger quorums with sizes ranging from 2 to 10, and measure the latency in completing the same 1,000 operations. Simultaneously, to gain a more in-depth understanding of the impact that each low-level instruction type has, we micro-benchmark the response time for all intra-system communications.

Results. Figure 4.7 plots the average response time for performing Key Generation, Decryption and Signing using IC quorums of different sizes. To begin with, Decryption is the fastest (119ms), since it implements a single round protocol. Moreover, when we compare the runtime between the single-IC run, and the runs with multiple ICs, we observe that the latency is very stable and the overhead remains always smaller than 0.8%. Hence, we conclude that the Decryption latency does not increase with the size of the quorum. This is due to the ICs performing the decryption computations simultaneously. It highlights that Decryption is only CPU bound, and the network capacity of our prototype does not affect it; and demonstrates that Myst can essentially provide increased assurance, with negligible impact on the decryption runtime. High-throughput decryption is of extreme importance in applications such as secure key derivation in mix network infrastructures [29], that heavily rely on decryption as compared to signing. The same is true for random number generation, since it also comprises of a single round protocol that can be executed in parallel.

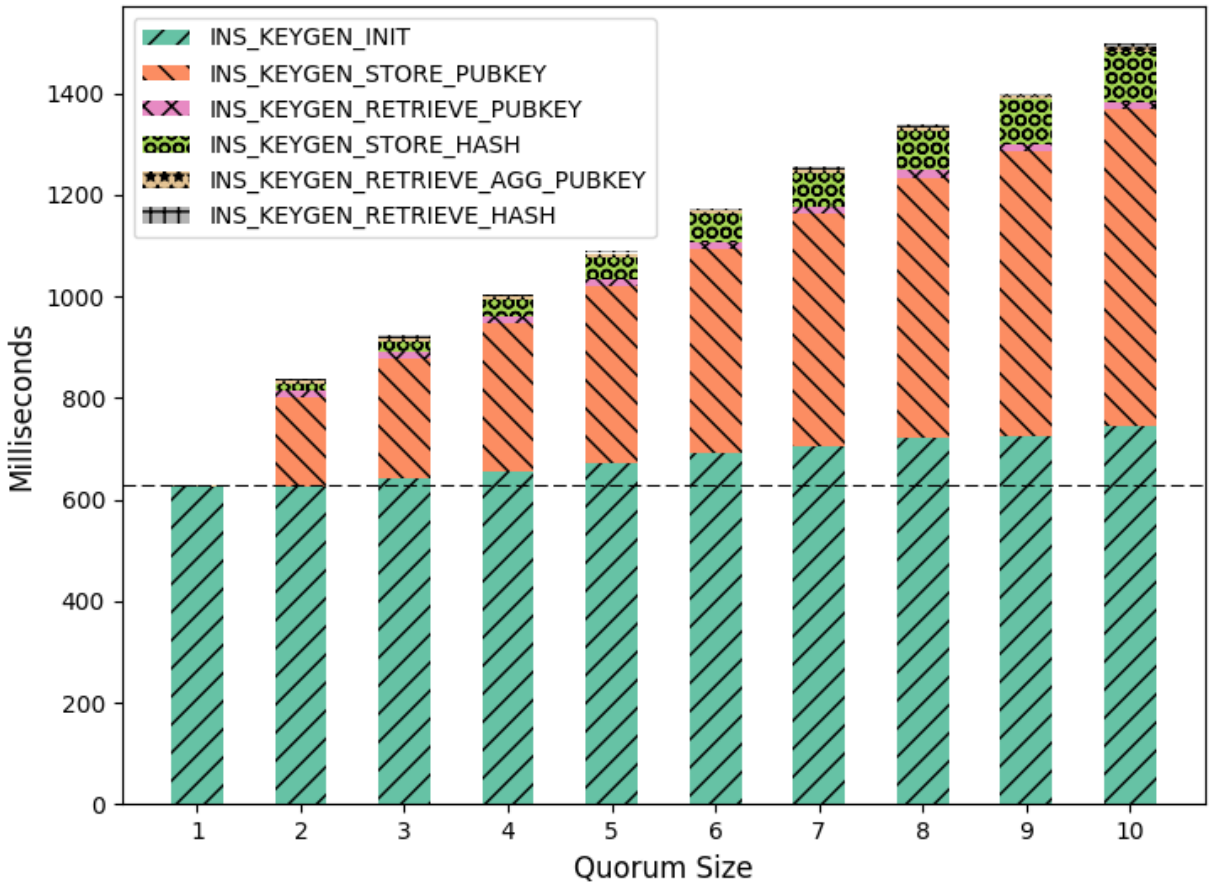

Figure 4.8: Breakdown of the runtime for low-level instructions that comprise the key generation operation, in relation to the quorum size. The reference horizontal line represent the cost of using a single IC.

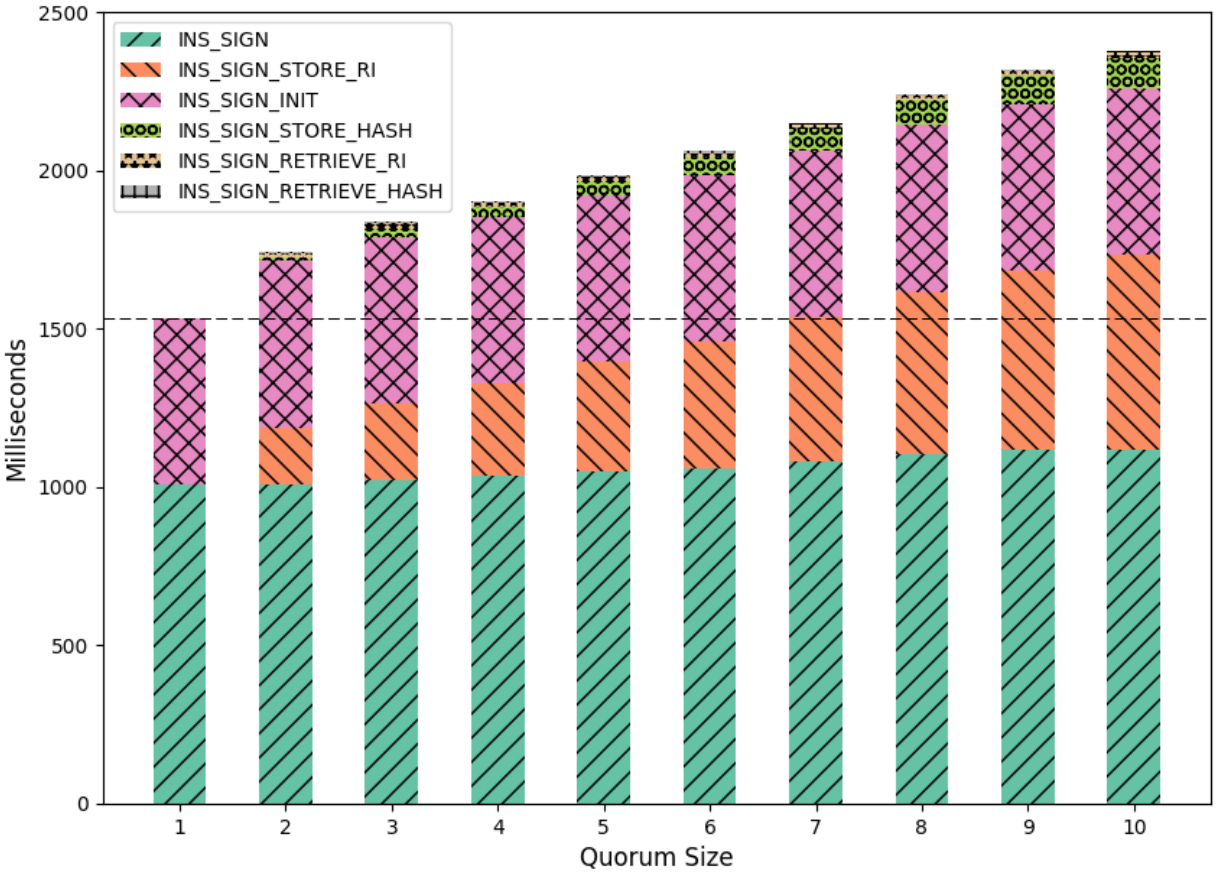

Figure 4.9: Breakdown of the runtime for low-level instructions that comprise the signing operation, in relation to the quorum size.

On the other hand, Key Generation (DKPG) requires two phases (commitment and revelation) and this adds significant latency. In particular, as seen in Figure 4.7 each additional IC results in an runtime increase of approximately 90ms. Figure 4.8 examines the timings of low-level operations involved in the key generation protocols. When quorums are small the cost of key generation is dominated by the "INS_KEYGEN_INIT" operation that generates a secret share, and derives a public key (624ms). However, as the IC quorums grow the operations related to exchanging public keys ("INS_KEYGEN_STORE_PUBKEY") and commitments ("INS_KEYGEN_STORE_HASH") become significant, and for a quorum of 10 ICs, nearly doubles the cost of key generation. However, for quorums of 3 the impact on runtime is merely 303ms, compared to a single IC. Other low-level operations have negligible cost, regardless of the quorum size.

Figure 4.9 examines the timings of the individual instructions used in the signing protocol. By far, the most expensive operation is "INS_SIGN" (1007ms), which is the operation that computes the signature share in each IC. As expected both the "INS_SIGN" operation and the "INS_SIGN_INIT" (527ms), used to generate the commitment share, take constant time and do not grow with the size of the quorum. On the other hand, as the quorums grow, the collective generation of the random commitment grows linearly. Similarly, with the key generation protocol, the instructions "INS_SIGN_STORE_PUBKEY" and "INS_SIGN_STORE_HASH" take most of the time. When comparing the single IC setup with a quorum of 3 ICs, we observe a 17% increase on the execution

time. However, this impact can be further reduced by pre-loading the commitment pairs. More specifically, the DKPG protocol can be modified so that the players generate and exchange a vector of pairs on each run. This will essentially reduce the corresponding part of the signing operation to single-IC levels. It should be noted that the admittedly low signing throughput of a single IC. However, we opted for JavaCards to provide the highest possible tampering-resistance, and be as diverse as possible.

4.6.2 Scalability & extensibility

Highly-scalable systems increase their capacity by simply adding more processing power. In our case, our hardware should be able to utilize more than one quorums at the same time, and distribute the computational load between them. Of course, we assume that all quorums are composed of multiple, diverse ICs, to ensure at least one IC per quorum is honest.

Methodology. To determine how efficiently our design scales in practice, we run a series of experiments that measure Myst's throughput, for varying numbers of processing quorums. As described in Section 4.5, our board supports up to 120 processing ICs which can be divided into multiple quorums and serve requests simultaneously. To benchmark the scalability of the system, on each iteration of the experiment we submit 10,000 requests for each high-level operation supported by our cryptosystem, and measure its throughput. However, this time we fix the quorum size k to 3, and on each iteration we increase the number of quorums serving the submitted requests by one, until we involve 40 quorums, which is the maximum number of 3-IC quorums that our prototype can support. For simplicity, we assign each processing IC to only one quorum. However, it is also technically possible for an IC to participate in more than one quorums, and store more than one secret shares.

Results. Figure 4.10 illustrates the throughput of the Myst system (in requests per second) as more of the ICs are used for processing transactions. The maximum throughput of Myst was 315ops/sec for Decryption and 21ops/sec for Signing, when using all 40 quorums. We also observe that as the number of quorums increases, the performance increases linearly, at a rate of ~ 9 requests per second per additional quorum for the Decryption operation, and 0.55 requests per second for Signing. This illustrates that the system is CPU bound, and the communication fabric between ICs and the host is not a bottleneck. Consequently, a system with our proposed architecture can be easily tailored for different use cases to provide the throughput needed in each of them.

4.6.3 Security considerations

Code & parameter provisioning is key to the security of Myst. If the code on all the ICs, or the cryptographic parameters is wrong, then the security of Myst will be compromised. We propose two strategies to ensure secure provisioning. First, we may assume that provisioning occurs at the factory. This leverages our assumption that some of the factories are honest, and ensures that an adversary would have to compromise all production to extract any secrets. The second strategy is to assume the existence of a secure off-line provisioning facility that is not under the control of the adversary. This facility need only be high-integrity, since no secrets are involved in the provisioning step (they are all generated within the ICs as part of later protocols).

Besides static provisioning of code, it may be interesting to offer the Myst architecture as a service – and we built a TCP interface to support such a use case. One may upload an application to the processor by opening a secure channel directly to the IC, and installing the application on a number of ICs. In order to separate applications from different authorities (to allow for

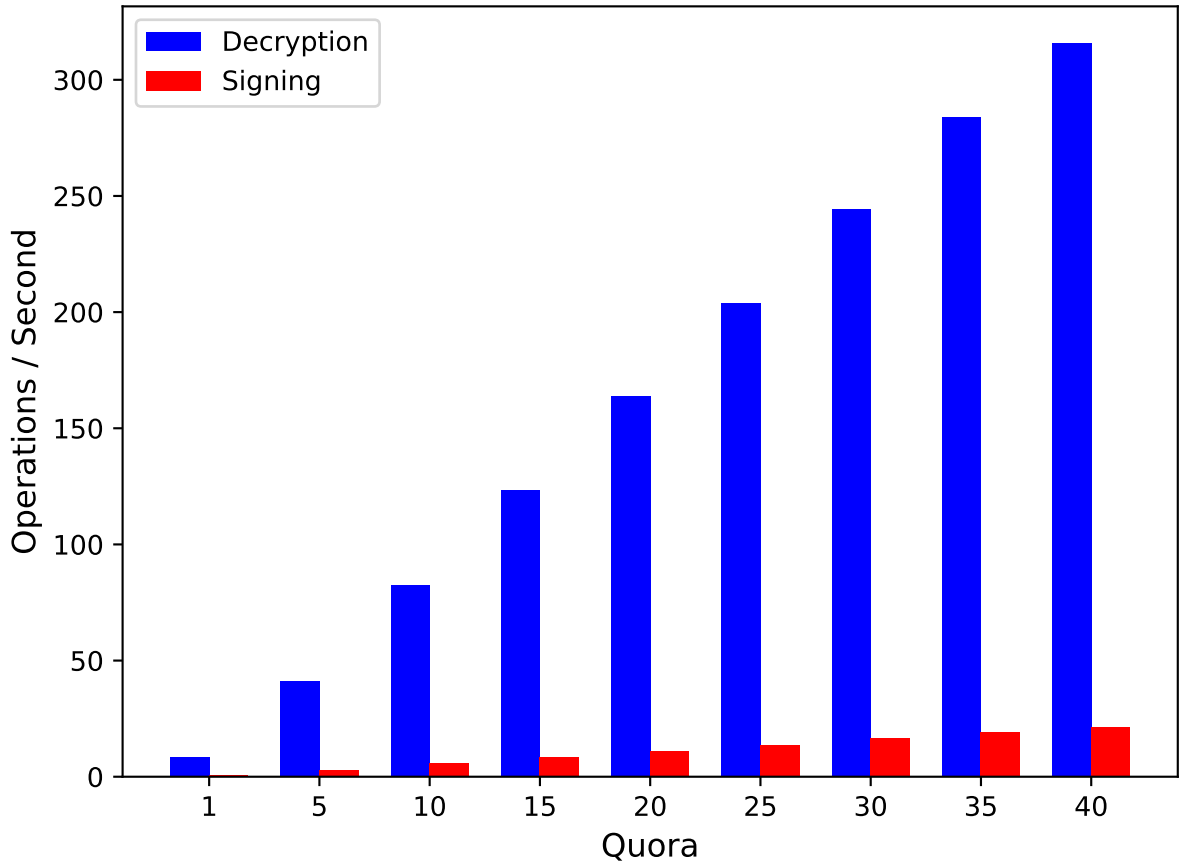

Figure 4.10: The average system throughput in relation to the number of quorums ($k = 3$) that serve requests simultaneously.

multiprogramming) we enforce an application identity to be set as a hash function of the public verification key of the application originator, and a application name of their choice. An application update, claiming a particular identity, will need a valid signature under the originator key before being loaded into the processor.

In all cases ICs need to be aware of their association with quorums. This association may be provided by the issuer of commands to maximize flexibility; or it may be statically determined at the time of provisioning.

SmartCard security. Smart Cards come with multiple benefits for our application as they were designed to operate in a hostile environment that is assumed to be fully controlled by the adversary [63]. For this reason, they come with tamper-resistance secure storage capabilities. Even so, a number of attacks against various cards have been introduced. For instance, there are multiple attacks that assume that the attacker is capable of tampering with the hardware during the provisioning phase [8, 32, 45, 56]. Moreover, a number of errors have been discovered in the realizations of the random number generators and other integral components of the cards [35, 25, 74]. However, SmartCards are still widely used, mainly because the low practicality of the great majority of the attacks, the constant evolution of anti-tampering techniques, and the very low incident count compared to the number of smartcards currently in use.

Assessment & parameterization. In Section 4.4, we highlighted the importance of the security threshold k, while in Section 4.6 we examined its relationship with other aspects of the system (e.g., performance). However, determining the exact threshold setting and the ICs participating in a quorum, given the lowest-acceptable bound for security is not a trivial task. Indeed, dependability and reliability assessment of a system has also been extensively studied by the fault-tolerance community [23, 51], before a consensus was reached. To solve this problem, the community promoted uniformity in assessments, by standardizing widely accepted terminology and formalizing definitions [52].

Drawing from these mature definitions, we also introduce two important factors that should be considered when setting the security threshold and determining the ICs participating in a quorum. To begin with, the assessment should consider the existence likelihood of hardware trojans or backdoors in the ICs given the supply chain they originate from. This can be further extended to account for the full supply chain, or for the steps where the vendor has no control over. Additionally, assuming that HT detection mechanisms are in place, their effectiveness should also be part of the decision. Obviously, standard quality assurance processes are not adequate for the detection of backdoors, but in cases where the vendor conducts more advanced inspections, this can increase its trust on specific ICs. However, the quantification of the above parameters remains an open problem, and as there is not commonly accepted way of evaluation, it largely depends on the information and testing capabilities each vendor has.

4.7 Related work

We provide here background and discuss related work in the literature of hardware trojans detection and prevention techniques. Moreover, we detail the relationship of our work with fault-tolerant designs and techniques.

Hardware trojans and countermeasures. Hardware Trojans are malicious modifications to the circuitry of a system component, and consist of a triggering unit, and a payload logic unit. The triggering unit simply monitors a set of inputs, for a specific sequence of events. Once the pre-determined sequence is observed, the payload logic is executed and performs a pre-defined set

of actions. The exact actions depend on the adversary's objectives, and may range from leaking stored secrets, to altering the functionality of the circuit. Due to the severity of the threat, there is an extended literature on fabrication-time trojan horses, introducing numerous attack classes and exploitation techniques. For instance, the authors in [49] design two hardware trojans and implement a proof-of-concept attack against the PRINCE cipher [19]. The novelty of their attacks is that they use dopant polarity changes (first introduced by [11]), to create a hard-to-detect fault-injection attack backdoor. Moreover, [59] also introduces a HT attacking RSA applications. In this attack, the adversary is able to use power supply fluctuations to trigger the trojan, which then leaks bits of the key through a signature. Another very hard to detect and devastating class of trojans was introduced by K. Yang et al. in [87] and is based on modifications in analog components. More specifically, instead of tampering with the processor in design time, they replaced hundreds of gates, with counter-based triggers, which given the right input, release a payload. Apart from these, detection evasion and stealthy triggering techniques have been proposed in [81, 80, 82, 50, 15].

Moreover, as discussed in Section 4.1, malicious components carrying trojans have been also observed in commercial and military hardware [68, 54, 46, 36, 67, 2, 53, 66] and a subset of those incidents also involved misbehaving cryptographic hardware [42, 64, 40, 65]. In all these cases, the errors were eventually attributed to honest design or fabrication mistakes, but the systems were left vulnerable to attacks regardless. For instance, one popular and well-studied example of attacks against weak cryptographic hardware is [13]. In this work, Bernstein et al. study the random number generators used in smartcards and found various malfunctioning pieces, that allowed him to break 184 public keys used in "Citizen Digital Certificates" by Taiwanese citizens.

To address the aforementioned threats two main directions have emerged: detection, and prevention. Detection techniques aim to determine whether any HTs exist in a given circuit and feature a wide range of approaches such as side-channel analysis [86, 84, 69, 3], logic testing [22], and trust verification [61, 7, 79, 91]. On the other hand, prevention techniques aim to either impede the introduction of HTs, or make HT easier to detect, such approaches are Split manufacturing [24, 83, 62], logic obfuscation [21] and runtime monitoring [44, 77]

Moreover, there also a smaller body of work which attempts to tackle the even harder problem of inferring additional information about the HT such as their triggers, payload, and exact location [86, 84].

Fault-tolerant systems. Component-redundancy and component-diversification are both key concepts used in N-variant systems that aim to achieve high fault-tolerance [23]. An example of such a system is the Triple-Triple Redundant 777 Primary Flight Computer [89, 90], that replicates the computations in three processors and then performs a majority voting to determine the final result. The applications of N-variance in security scenarios has been studied in only few works aiming to protect systems against software attacks. In particular, [26] introduces a method to generate randomized system copies, that will have disjoint exploitation sets, thus achieving memory safety. On the other hand, [4] proposes a N-variant method for IC diversification, aiming again to achieve disjoint exploitation sets. However, this method is not effective against fabrication-time attacks launched during the IC manufacturing. Finally, heterogeneous architectures with COTS components have been also proposed in [9, 10]. However, the practicality of these works is very limited as the computations are simply replicated between the different components, thus protecting against only a small subset of existing hardware attacks.

4.8 Conclusion

The literature on hardware trojans and malicious circuitry extends over ten years, and includes an abundance of trojan designs, mitigation techniques and countermeasure evasion methods. This line of work assumes that due to the arms race, between trojan horses and mitigation techniques, countermeasures will never be 100% effective against all possible threats. To resolve this, we introduce Myst, which brings the adversary into the unfavorable position of having to achieve 100% compromise to gain any advantage. Specifically, by employing threshold schemes and a redundancy-based architecture, Myst is able to distribute both secrets and computations among multiple, diverse integrated circuits. Consequently, an adversary aiming to breach the confidentiality or the integrity of the system, must be able to compromise all the ICs. This is not a trivial task when ICs are sourced through non-crossing supply chains. To evaluate Myst, we build a custom board featuring 120 Smart Cards controlled by an Artix-7 FPGA. The maximum throughput for distributed decryption is 315ops/sec and the operation comes with an overhead less than 0.8% compared a non-distributed system. Moreover, the more complex signing operation has a 17% impact on runtime. All in all, our results show that Myst is highly scalable, and provides strong guarantees for the security of the system at a reasonable performance cost.

Joint Research with Enigmabridge. To provide a reliable infrastructure for PANORAMIX nodes, we looked into hardware solutions that enable secure key generation and storage, as well as other necessary cryptographic operations (i.e., decryption, signing). Such features are commonly found in commercially available hardware security modules (HSMs). However, the threat model commercial HSMs operate under is different than the one PANORAMIX nodes were designed for. As a result, while HSMs can rely on a single trusted vendor to provide secure hardware components, PANORAMIX assumes a much stronger adversary that can potentially compromise the hardware supply chain of some of the components.

For this purpose, we designed a novel architecture for cryptographic hardware that retains its security properties even if some of its components are actively malicious. To implement, evaluate and deploy our design, we researched jointly with experts from Enigmabridge (20 Bridge St, Cambridge CB2 1UF, UK). Enigmabridge experts produced the hardware schematics and built the circuit boards; UCL programmed the integrated circuits of the board with firmware that implements the protocols realizing the required functionality (i.e., key generation, decryption, signing). We jointly performed several testing rounds to ensure both the completeness and the robustness of the final product.

The outcomes of this cooperation are public. The details of the architecture, the hardware components used in the board, the cryptographic protocols and our source code are freely available online (under MIT license, `https://github.com/OpenCryptoProject/Myst/`), and have already been discussed in various public presentations of our work and in our recently published paper. The website of the project is: `https://BackdoorTolerance.org` which we will keep updating and enriching with more info.

Bibliography

[1] M. Adalier. Efficient and secure elliptic curve cryptography implementation of curve p-256.

[2] S. Adee. The hunt for the kill switch. *IEEE Spectrum*, 45(5):34–39, 2008.

[3] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan detection using ic fingerprinting. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 296–310. IEEE, 2007.

[4] Y. Alkabani and F. Koushanfar. N-variant IC design: methodology and applications. In *Proceedings of the 45th Design Automation Conference, DAC 2008, Anaheim, CA, USA, June 8-13, 2008*, pages 546–551, 2008.

[5] J. Appelbaum, J. Horchert, and C. Stöcker. Shopping for spy gear: Catalog advertises nsa toolbox. *Der Spiegel*, 29, 2013.

[6] M. Backes, M. Dürmuth, and D. Unruh. Compromising reflections-or-how to read lcd monitors around the corner. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 158–169. IEEE, 2008.

[7] C. Bao, Y. Xie, and A. Srivastava. A security-aware design scheme for better hardware trojan detection sensitivity. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2015, Washington, DC, USA, 5-7 May, 2015*, pages 52–55, 2015.

[8] G. Barbu, H. Thiebeauld, and V. Guerin. Attacks on java card 3.0 combining fault and logical attacks. In *International Conference on Smart Card Research and Advanced Applications*, pages 148–163. Springer, 2010.

[9] M. Beaumont, B. Hopkins, and T. Newby. Safer path: Security architecture using fragmented execution and replication for protection against trojaned hardware. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1000–1005. EDA Consortium, 2012.

[10] M. Beaumont, B. Hopkins, and T. Newby. Hardware trojan resistant computation using heterogeneous cots processors. In *Proceedings of the Thirty-Sixth Australasian Computer Science Conference-Volume 135*, pages 97–106. Australian Computer Society, Inc., 2013.

[11] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson. Stealthy dopant-level hardware trojans. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 197–214. Springer, 2013.

[12] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson. Stealthy dopant-level hardware trojans: extended version. *J. Cryptographic Engineering*, 4(1):19–31, 2014.

[13] D. J. Bernstein, Y.-A. Chang, C.-M. Cheng, L.-P. Chou, N. Heninger, T. Lange, and N. Van Someren. Factoring rsa keys from certified smart cards: Coppersmith in the wild. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 341–360. Springer, 2013.

[14] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak sponge function family main document. *Submission to NIST (Round 2)*, 3:30, 2009.

[15] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan. Hardware trojan attacks: Threat analysis and countermeasures. *Proceedings of the IEEE*, 102(8):1229–1247, 2014.

[16] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan. Hardware trojan attacks: threat analysis and countermeasures. *Proceedings of the IEEE*, 102(8):1229–1247, 2014.

[17] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 103–112. ACM, 1988.

[18] D. Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*, pages 48–63. Springer, 1998.

[19] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, et al. Prince–a low-latency block cipher for pervasive computing applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 208–225. Springer, 2012.

[20] F. Brandt. Efficient cryptographic protocol design based on distributed el gamal encryption. In D. Won and S. Kim, editors, *Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers*, volume 3935 of *Lecture Notes in Computer Science*, pages 32–47. Springer, 2005.

[21] R. S. Chakraborty and S. Bhunia. Security against hardware trojan through a novel application of design obfuscation. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, pages 113–116. ACM, 2009.

[22] R. S. Chakraborty, F. G. Wolff, S. Paul, C. A. Papachristou, and S. Bhunia. MERO: A statistical approach for hardware trojan detection. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, pages 396–410, 2009.

[23] L. Chen and A. Avizienis. N-version programming: A fault-tolerance approach to reliability of software operation. In *Digest of Papers FTCS-8: Eighth Annual International Conference on Fault Tolerant Computing*, pages 3–9, 1978.

[24] Z. Chen, P. Zhou, T. Y. Ho, and Y. Jin. How secure is split manufacturing in preventing hardware trojan? In *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*, pages 1–6, Dec 2016.

[25] N. T. Courtois. The dark side of security by obscurity and cloning mifare classic rail and building passes, anywhere, anytime. 2009.

[26] B. Cox and D. Evans. N-variant systems: A secretless framework for security through diversity. In *Proceedings of the 15th USENIX Security Symposium, Vancouver, BC, Canada, July 31 - August 4, 2006*, 2006.

[27] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, pages 1–18, 2013.

[28] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[29] G. Danezis, C. Diaz, and P. Syverson. Systems for anonymous communication. *Handbook of Financial Cryptography and Security, Cryptography and Network Security Series*, pages 341–389, 2009.

[30] D. Edenfeld, A. B. Kahng, M. Rodgers, and Y. Zorian. 2003 technology roadmap for semiconductors. *IEEE Computer*, 37(1):47–56, 2004.

[31] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[32] M. Farhadi and J.-L. Lanet. Chronicle of a java card death. *Journal of Computer Virology and Hacking Techniques*, pages 1–15, 2016.

[33] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.

[34] T. Force. High performance microchip supply. 2005.

[35] B. Fredriksson. A case study in smartcard security analysing mifare classic rev. 2016.

[36] S. Gallagher. Photos of an nsa upgrade factory show cisco router getting implant. *Ars Technica*, 14, 2014.

[37] D. Geer, R. Bace, P. Gutmann, P. Metzger, C. P. Pfleeger, J. S. Quarterman, and B. Schneier. Cyberinsecurity: The cost of monopoly. *How the dominance of Microsofts products poses a risk to society*, 2003.

[38] D. Genkin, A. Shamir, and E. Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. In *International Cryptology Conference*, pages 444–461. Springer, 2014.

[39] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptology*, 20(1):51–83, 2007.

[40] D. Goodin. 'we cannot trust intel and vias chip-based crypto', freebsd developers say. http://arstechnica.com/security/2013/12/we-cannot-trust-intel-and-vias-chip-based-crypto-freebsd-developers-say/, December 2013.

[41] T. Granlund and P. L. Montgomery. Division by invariant integers using multiplication. In *ACM SIGPLAN Notices*, volume 29, pages 61–72. ACM, 1994.

[42] F. S. W. Group. Freebsd developer summit: Security working group. `https://wiki.freebsd.org/201309DevSummit/Security`, December 2013.

[43] S. Heck, S. Kaza, and D. Pinner. Creating value in the semiconductor industry. *McKinsey & Company*, 2011.

[44] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 159–172, 2010.

[45] J. Iguchi-Cartigny and J.-L. Lanet. Developing a trojan applets in a smart card. *Journal in computer virology*, 6(4):343–351, 2010.

[46] Y. Jin and Y. Makris. Hardware trojans in wireless cryptographic ics. *IEEE Design & Test of Computers*, 27(1), 2010.

[47] R. Johnson et al. Introduction to the spring framework. *TheServerSide. com*, 21:22, 2005.

[48] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou. Designing and implementing malicious hardware. In *First USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '08, San Francisco, CA, USA, April 15, 2008, Proceedings*, 2008.

[49] R. Kumar, P. Jovanovic, W. P. Burleson, and I. Polian. Parametric trojans for fault-injection attacks on cryptographic hardware. *IACR Cryptology ePrint Archive*, 2014:783, 2014.

[50] S. Kutzner, A. Y. Poschmann, and M. Stöttinger. Hardware trojan design and detection: a practical evaluation. In *Proceedings of the Workshop on Embedded Systems Security, WESS 2013, Montreal, Quebec, Canada, September 29 - October 4, 2013*, pages 1:1–1:9, 2013.

[51] J.-C. Laprie. Dependable computing and fault-tolerance. *Digest of Papers FTCS-15*, pages 2–11, 1985.

[52] P. A. Lee and T. Anderson. *Fault tolerance: principles and practice*, volume 3. Springer Science & Business Media, 2012.

[53] J. Markoff. Old trick threatens the newest weapons. *The New York Times*, 27, 2009.

[54] S. Mitra, H.-S. P. Wong, and S. Wong. The trojan-proof chip. *IEEE Spectrum*, 52(2):46–51, 2015.

[55] I. Miyamoto, T. H. Holzer, and S. Sarkani. Why a counterfeit risk avoidance strategy fails. *Computers & Security*, 2017.

[56] W. Mostowski and E. Poll. Malicious code on java card smartcards: Attacks and countermeasures. In *International Conference on Smart Card Research and Advanced Applications*, pages 1–16. Springer, 2008.

[57] U. S. D. S. B. T. F. on High Performance Microchip Supply. *Defense science board task force on high performance microchip supply*. Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics, 2005.

[58] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, pages 129–140, 1991.

[59] A. Pellegrini, V. Bertacco, and T. Austin. Fault-based attack of rsa authentication. In *Proceedings of the conference on Design, automation and test in Europe*, pages 855–860. European Design and Automation Association, 2010.

[60] M. Potkonjak, A. Nahapetian, M. Nelson, and T. Massey. Hardware trojan horse detection using gate-level characterization. In *Proceedings of the 46th Design Automation Conference, DAC 2009, San Francisco, CA, USA, July 26-31, 2009*, pages 688–693, 2009.

[61] J. J. Rajendran and S. Garg. Logic encryption. In *Hardware Protection through Obfuscation*, pages 71–88. Springer, 2017.

[62] J. J. Rajendran, O. Sinanoglu, and R. Karri. Is split manufacturing secure? In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1259–1264. EDA Consortium, 2013.

[63] W. Rankl and W. Effing. *Smart card handbook*. John Wiley & Sons, 2004.

[64] RT. 'we cannot trust them anymore': Engineers abandon encryption chips after snowden leaks. `https://www.rt.com/usa/snowden-leak-rng-randomness-019/`, December 2013.

[65] B. Schneier. Surreptitiously tampering with computer chips. `https://www.schneier.com/blog/archives/2013/09/surreptitiously.html`, November 2013.

[66] T. Shrimpton and R. S. Terashima. A provable-security analysis of intels secure key rng. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 77–100. Springer, 2015.

[67] S. Skorobogatov. Hardware assurance and its importance to national security. *Available Online:¡ http://www. cl. cam. ac. uk/sps32/secnews. html*, 2012.

[68] S. Skorobogatov and C. Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, pages 23–40, 2012.

[69] O. Soll, T. Korak, M. Muehlberghuber, and M. Hutter. Em-based detection of hardware trojans on fpgas. In *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, pages 84–87. IEEE, 2014.

[70] M. Stamp. Risks of monoculture. *Commun. ACM*, 47(3):120, 2004.

[71] D. R. Stinson and R. Strobl. Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates. In V. Varadharajan and Y. Mu, editors, *Information Security and Privacy, 6th Australasian Conference, ACISP 2001, Sydney, Australia, July 11-13, 2001, Proceedings*, volume 2119 of *Lecture Notes in Computer Science*, pages 417–434. Springer, 2001.

[72] L. Strigini. Fault tolerance against design faults. 2005.

[73] P. Švenda. Nuances of the javacard api on the cryptographic smart cards–jcalgtest project.

[74] P. Svenda, M. Nemec, P. Sekan, R. Kvasnovsky, D. Formanek, D. Komarek, and V. Matyas. The million-key question – investigating the origins of rsa public keys. In *The 25th USENIX Security Symposium (UsenixSec'2016)*, pages 893–910. USENIX, 2016.

[75] M. Tehranipoor and C. Wang. *Introduction to hardware security and trust*. Springer Science & Business Media, 2011.

[76] S. Turner, R. Housley, T. Polk, D. R. Brown, and K. Yiu. Elliptic curve cryptography subject public key information. 2009.

[77] A. Waksman and S. Sethumadhavan. Tamper evident microprocessors. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 173–188. IEEE, 2010.

[78] A. Waksman and S. Sethumadhavan. Silencing hardware backdoors. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 49–63. IEEE, 2011.

[79] A. Waksman, M. Suozzo, and S. Sethumadhavan. FANCI: identification of stealthy malicious logic using boolean functional analysis. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 697–708, 2013.

[80] X. Wang. *Hardware trojan attacks: Threat analysis and low-cost countermeasures through golden-free detection and secure design.* PhD thesis, Case Western Reserve University, 2014.

[81] X. Wang, T. Mal-Sarkar, A. R. Krishna, S. Narasimhan, and S. Bhunia. Software exploitable hardware trojans in embedded processor. In *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2012, Austin, TX, USA, October 3-5, 2012*, pages 55–58, 2012.

[82] X. Wang, S. Narasimhan, A. Krishna, T. Mal-Sarkar, and S. Bhunia. Sequential hardware trojan: Side-channel aware design and placement. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 297–300. IEEE, 2011.

[83] Y. Wang, P. Chen, J. Hu, and J. Rajendran. The cat and mouse in split manufacturing. In *Proceedings of the 53rd Annual Design Automation Conference, DAC 2016, Austin, TX, USA, June 5-9, 2016*, pages 165:1–165:6, 2016.

[84] S. Wei and M. Potkonjak. Scalable hardware trojan diagnosis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(6):1049–1057, June 2012.

[85] S. Wei and M. Potkonjak. The undetectable and unprovable hardware trojan horse. In *Proceedings of the 50th Annual Design Automation Conference*, page 144. ACM, 2013.

[86] S. Wei and M. Potkonjak. Self-consistency and consistency-based detection and diagnosis of malicious circuitry. *IEEE Trans. VLSI Syst.*, 22(9):1845–1853, 2014.

[87] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester. A2: Analog malicious hardware, 2016.

[88] A. Yeh. Trends in the global ic design service market. *DIGITIMES research*, 2012.

[89] Y. C. Yeh. Triple-triple redundant 777 primary flight computer. In *Aerospace Applications Conference, 1996. Proceedings., 1996 IEEE*, volume 1, pages 293–307. IEEE, 1996.

[90] Y. C. Yeh. Design considerations in boeing 777 fly-by-wire computers. In *High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International*, pages 64–72. IEEE, 1998.

[91] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu. Veritrust: Verification for hardware trust. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(7):1148–1161, 2015.

[92] Y. Zhang, H. M. Vin, L. Alvisi, W. Lee, and S. K. Dao. Heterogeneous networking: a new survivability paradigm. In *Proceedings of the New Security Paradigms Workshop 2001, Cloudcroft, New Mexico, USA, September 10-13, 2001*, pages 33–39, 2001.

Part III

Robust security definitions for mix networks

5. Computing tight anonymity bounds for Tor against malicious network infrastructure

The benefits of our interconnected online world, although indisputable, are accompanied by threats to user's privacy of unprecedented magnitude. Activities of citizens are constantly tracked and profiled both by the advertising industry and by governmental agencies. As a result, public interest in privacy enhancing techniques and anonymous communication systems has increased significantly over the course of the last few years. Millions of users now use anonymizing proxies, VPNs and in particular the Tor network [5, 6], to anonymously browse the web. The utility of such tools, however, is directly related to the degree of anonymity they provide and measuring this degree of anonymity is an active field of research.

In this chapter, we present the first rigorous methodology for formally quantifying the reduction of anonymity in an anonymous communication protocol against malicious network infrastructures, including combinations of autonomous systems, Internet exchange points, and submarine cables. Deriving rigorous and under reasonable assumptions tight anonymity guarantees requires a well-defined foundation, including robust anonymity definitions for anonymous communication systems. Thus, we leverage the AnoA [12] definition of anonymity that is inspired by differential privacy for our quantification. Finally, we perform an evaluation of Tor's anonymity against network eavesdroppers.

Relation to the goals of PANORAMIX

Our methodology directly targets task 3.3 and makes significant contributions our goal of analysing the interplay between anonymous communication systems and differential privacy. First, we define and calculate differential privacy like guarantees for Tor, one of the most widely used instantiations of an anonymous channel, if parts of the network infrastructure are controlled by an adversary. We achieve strong, composable and within our reasonable abstractions tight results against malicious network infrastructure. Our definition of anonymity and our methodology can be modularly adapted and applied to other anonymous communication protocols, such as classical mixes.

In our large-scale evaluation we consider the anonymity impact of real companies and surveillance performed at real physical locations on Tor. To this end, we analyse anonymity against companies providing very significant autonomous systems (Level 3, NTT, DTAG), as well as of the largest European Internet exchange point (DE-CIX) and the landing point of submarine cables connecting the US and England near Bude, in England. We compare results for Tor with LASTor, a variant of Tor proposed to counter malicious or overly curious Internet infrastructure and we find that both are vulnerable against our adversaries. Our analysis establishes a base-line for our own

instantiations of anonymous channels, i.e., mix network based protocols.

Concretely, we make the following contributions: First, we provide an algorithmic approach for computing Tor's *reduction of anonymity* when facing malicious network infrastructure. The obtained bounds are shown to be correct for the cryptographic realization of Tor using a recently introduced AnoA framework for reasoning about anonymous communication systems. Consequently, the bounds follow the intuition and definitions of differential privacy and come with the composability and post-computation resilience known from differential privacy as well. We show that the bounds are tight under reasonable assumptions.

Second, we *evaluated* our approach against several adversarial settings that rely on observing parts of Tor's Internet structure. Our evaluation shows that Tor is highly susceptible to observations by a top-tier provider such as Level 3, NTT, and DTAG, exhibiting a reduction of anonymity of more than 20%, for NTT and Level 3 respectively. Our analysis marks a significant step towards understanding the anonymity provided by Tor. This step is necessary for safely using Tor as a component of larger privacy preserving systems.

5.1 Motivation

The Internet has evolved from a mere communication network used by millions of users to a global platform for social networking, communication, education, entertainment, trade, and political activism used by billions of users. In addition to the indisputable societal benefits of this transformation, the mass reach of the Internet has created new powerful threats to online privacy. Today, users are constantly tracked, monitored, and profiled, both with the intent of monetizing their personal information through targeted advertisements, and by nearly omnipotent agencies that rely on the mass collection of metadata for conducting dragnet surveillance at the planetary scale.

Millions of users try to escape this profiling by relying on the anonymous communication network Tor, which promises to hide their identities and their communication partners [1, 5, 6, 17]. In the Tor network, users connect to a series of three anonymizing proxies successively, which constitute a so-called *Tor circuit*. This circuit ensures, using appropriate cryptographic protection, that each proxy only learns its predecessor and its successor, and thereby hides both the user's and the recipient's IP addresses. Tor has become a core building block for various privacy-enhancing technologies, e.g., in the Tor Browser Bundle [6] and in the privacy-preserving live operating system Tails [4]. Recently, Mozilla has announced a collaboration with Tor to accelerate practical advances in privacy technology for the Web [31].

One of the most important research questions concerning Tor is to rigorously determine and quantify the degree of anonymity that Tor provides against various adversarial abilities. Amongst these abilities, traffic correlation is widely considered as the by-far most successful and realistic attack to undermine the anonymity of Tor. In traffic correlation, an adversary that is capable of monitoring traffic at different connections (e.g., between the sender and the guard node, and between the exit node and the recipient) uses traffic analysis techniques to identify and link common patterns, and to thereby deanonymize Tor users and their communicating partners. Recent studies have shown the impressive potential of this attack strategy [26, 18, 19], and current solutions to counter traffic correlation attacks still impose prohibitively high performance overheads for low-latency communications such as Tor.

Mounting traffic correlation attacks crucially relies on the ability to monitor traffic at different, suitable connections. In reality, this corresponds to getting access, or to subverting, corresponding parts of the network infrastructure. Recent studies show that the set of potential autonomous

systems, which constitute a core part of this infrastructure, that are capable of using traffic correlation is presumably much larger than expected, as a result of Tor's routing characteristics and intentional attempts to manipulate the Internet's routing system [35, 34]. It is hence expected that traffic correlation attacks – and thus: compromised network infrastructure – is the currently most impeding factor for the anonymity offered by the Tor network. This expectation is further substantiated by first empirical studies that aimed at determining the vulnerability of Tor against corrupted autonomous systems and corresponding traffic correlation attacks [26, 33]; we refer to the related work section for more details.

However, while such approaches provide first empirical observations of a user's anonymity using simulation and subsequent extrapolations, no work exists thus far that establishes semantically well-founded *bounds* to which extent anonymity is being reduced if specific parts of the network infrastructure are considered to be under adversarial control. In addition to deriving quantitative anonymity bounds for Tor, this would shed light on related, largely unexplored questions of practical relevance, e.g., how the mapping between the Tor overlay network and the actual Internet topology influences anonymity guarantees, and which parts of the network infrastructure are particularly valuable targets for adversarial observations.

5.1.1 Our contributions

In this chapter, we present the first rigorous methodology for formally quantifying the reduction of anonymity in Tor against malicious network infrastructure, including (combinations of) autonomous systems (AS), Internet exchange points (IXP) and submarine cables. Concretely, this chapter makes the following contributions: (theory) an algorithmic approach for computing Tor's *reduction of anonymity* when facing *malicious network infrastructure*; and (practice) an *evaluation* how anonymity bounds for Tor are formally affected by various malicious infrastructure settings.

Computing reduction of anonymity. Formally quantifying the reduction of anonymity against adversaries that control parts of the network infrastructure imposes several challenges. Most importantly, since we strive for rigorous bounds instead of only simulating possible runs and extrapolating the results, we have to accurately compute the degree and the corresponding reduction of anonymity, given Tor's specification, the current status of the Tor overlay network, Tor's Internet topology, the anonymity notion under consideration, and an accurately modelled adversary that reflects the corrupted infrastructure. In particular, the computation of the bounds requires to carefully consider all potential adversarial observations and to assess their impact on a user's anonymity. To this end, we provide an algorithm that computes the reduction of anonymity based on the anonymity notion under consideration and the adversary's possible observations that are derived from the considered network structure. We then formally show that the obtained bounds correspond precisely to the notion of optimal adversary advantage against an idealized version of Tor in the AnoA framework [12] – a recent framework for computing quantitative bounds for anonymous communication systems. Using this embedding into AnoA, we show that our bounds hold as well for the cryptographic realization of Tor and that they are tight under the assumption that perfect traffic correlation is possible (both up to a negligible factor). This allows for computing precise bounds in the presence of malicious network infrastructure, and hence supersedes existing approaches that only heuristically derive quantitative anonymity statements.

Implementation and evaluation. Finally, we evaluated our methodology against several adversarial settings that rely on observing parts of the Internet topology, in particular various AS-level adversaries (Level 3, NTT, DTAG), and a DE-CIX adversary, and a submarine cable adversary. We first constructed a model of the Internet topology on which Tor relies, which we call Tor's

Internet topology. Then, we ran our computations for the reduction of anonymity on our snapshot of Tor's Internet topology and the corresponding Tor consensus, taking into account 100 sampled combinations of senders and recipients. Using these experiments, we compare Tor's path selection algorithm with the LASTor path selection algorithm [8], since LASTor was specifically designed to reduce the impeding effects of malicious infrastructure by choosing geographically close Tor nodes. Our experiments in particular show that both path selection algorithms are highly susceptible to an observing top-tier provider, facing a reduction of anonymity of up to 27.8% for Level 3. A collusion of Level 3, NTT and DTAG even reduces anonymity by 47.2%.

5.1.2 Outline

In Section 5.2, we provide our algorithmic approach for computing anonymity bounds in the presence of malicious network infrastructure. In Section 5.3, we show that these bounds are accurate for Tor, and that they are tight under the assumption that perfect traffic correlation is possible. Section 5.4 contains our evaluation results for various settings of malicious infrastructure. Section 5.5 reviews further related work.

5.2 Computing anonymity bounds

In this section, we provide an algorithmic approach for computing anonymity bounds for Tor in the presence of an adversary that controls selected parts of the Internet infrastructure. We first characterize all possible observation points that an adversary can observe in Tor and introduce the anonymity notions considered in this chapter (Section 5.2.1). Based on this characterization and these notions, we then show how to compute the reduction of anonymity based on any given model of corrupted components of the Internet's topology (Section 5.2.2). We will later instantiate this algorithm with Tor's Internet topology and thereby obtain concrete real-life anonymity bounds for various adversarial settings.

In this section, we will concentrate on the algorithmic aspects of computing the reduction of anonymity, and use the terms *anonymity notions*, *observation points*, and *adversary's advantage* that contribute to the overall anonymity assessment only in an informal way. We will give a formal semantics to those terms in Section 5.3, and then rigorously prove the correctness and tightness (under some assumptions) of our computed bounds based on this semantics.

5.2.1 Observation points and anonymity notions

Every Tor circuit contains four different sub-paths that an adversary could observe by corrupting suitable parts of the network infrastructure, see the upper part of Figure 5.1: between the sender S_0 and the guard node G (denoted as S_0-G), between the guard node and the middle node M (denoted G-M), between the middle node and the exit node X (denoted M-X), and between the exit node and the recipient R_0 (denoted X-R_0). Observing each of these sub-paths enables the adversary to draw conclusions about the sender and/or the recipient of the circuit, and to thereby reduce their degree of anonymity.

Some of these conclusions are straightforward and absolute: if the connection between the sender and the guard node can be observed, the sender is trivially deanonymized as the origin of the communication. Similar, the ability to observe the connection between the exit node and the recipient unveals the recipient as the destination of the communication.

While existing papers that strive to assess anonymity are limited to these observations, additional conclusions can be drawn when corrupting *any* node, even the seamingly less threating

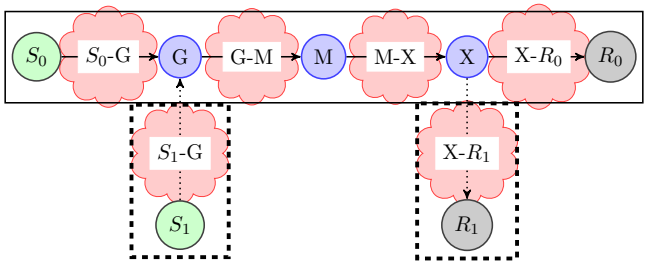

Figure 5.1: Upper part: Observation points of a network-observing adversary for a given Tor circuit: sender to guard node (S_0-G), guard node to middle node (G-M), middle node to end node (M-X), and end note to recipient (X-R_0). Lower part: Further observation points for an alternative sender S_1 (S_1-G) and for an alternative recipient R_1 (X-R_1) that will be used to define different anonymity notions.

middle nodes. These conclusions are more subtle but no less influencial. For instance, if the adversary observes or knows that the sender communicates over a specific port, all exit nodes can be excluded that do not support this port choice, and hence any communication that involves excluded exit nodes cannot originate from that sender. Moreover, excluding exit nodes influences the probability which nodes are being selecting as guard or middle node in this circuit by Tor's path selection algorithm (the selection takes so-called family relationships and further constraints into account, see Section 5.4.2). Technically, this means that the a-priori probability distribution over circuits induced by Tor's path selection algorithm is now replaced by an a-posteriori distribution that is conditioned on the network observations of the adversary. This enables the adversary to draw further conclusions and to thereby reduce anonymity.

To reflect all these capabilities, we model the knowledge an adversary gains from its observations by comparing its ability to distinguish two *scenarios* that differ in their involved senders and recipients. This follows the established concept of indistinguishability-based definitions in cryptography (e.g., IND-CCA secure encryption): One of these two scenarios is selected at random, a Tor circuit is created for this scenario, and the adversary is then allowed to make observations for this circuit depending on the considered network corruption model. The adversary knows the set-up of both scenarios, its observations, and it then has to decide which scenario it currently observes. The reduction of anonymity is then defined as the *adversary's advantage*, i.e., as the probability of correctly telling both scenarios apart.

We consider three commonly considered notions of communication anonymity in this chapter: *sender anonymity* (i.e., determine who is sending a message), *recipient anonymity* (i.e., determine to whom a message is sent), and *relationship anonymity* (i.e., determine a correlation between senders and recipients). Each of these three notions requires its own two scenarios to define the adversary's advantage with respect to this notion. This is illustrated in Figure 5.1: for sender anonymity, an additional sender S_1 is considered, i.e., the two scenarios differ in the sender, but share the same recipient R_0. Similarly, an additional recipient R_1 is considered for recipient anonymity, and the two scenarios differ in the recipient, but share the same sender S_0. Capturing the absence of correlations to define relationship anonymity is more involved. We both consider an additional sender S_1 and an additional recipient R_1: The first relationship anonymity scenario considers the two cases that S_0 communicates with R_0 and that S_1 communicates with R_1; the second scenario considers the communication from S_0 to R_1, and from S_1 to R_0. After the scenario has been selected, one of the two described cases for this scenario is selected at random, then a Tor circuit is created for this case, and the adversary can start making its observations for this circuit.

For each anonymity notion, the adversary can hence make network observations at the following *observation points* in order to tell apart the scenarios. For each anonymity notion, the four sub-paths S_0-G, G-M, M-X, and X-R_0 are canonical observation points as described earlier in this section. For sender anonymity, an additional observation point is S_1-G, and similarly X-R_1 for recipient anonymity. For relationship anonymity, both S_1-G and X-R_1 are additional observation points. In all these scenarios, we additionally consider the absence of an observed communication as a distiguished observation, denoted $\perp$, since realizing that a communication has not happened at the observated network path constitutes an observation in itself.

We will show in Section 5.3 that the advantage of an eavesdropping adversary can be characterized by the sum over the probability differences of the observations on each of these sub-paths. We stress that a network-observing adversary does not learn anything from the encrypted content of Tor traffic in our model (except for the ability to correlate it with other Tor traffic). We will show that this assumption is sound by grounding our formal analysis on a faithful idealization of Tor [11].

5.2.2 Calculating the reduction of anonymity

The algorithm for computing the reduction of anonymity (Figure 5.2) works in two phases: After initializing the underlying `store` data structure, it first computes the probability of every possible observation over the probabilistically generated Tor circuits for all four possible sender-recipient pairs.[1] We call this the *observation phase*. In the second step, we derive the advantage of the adversary by suitably aggregating the probabilities of the respective observations depending on the considered anonymity notion. We call this the *deduction phase*.

To improve the performance of our algorithm, we compute guarantees for all three anonymity notions at once, and hence simultaneously consider all four different sender-recipient pairs (and discard the two respective obsolete pairs for sender and recipient anonymity, see below). We now describe the individual phases in more detail.

Underlying notions. Our algorithm centrally relies on the notion of observations and of malicious infrastructure. For the set of senders $\mathcal{S}$, the set of Tor nodes $\mathcal{N}$, and the set of recipients $\mathcal{R}$, let $\mathcal{S}_\perp := \mathcal{S} \cup \{\perp\}$, $\mathcal{N}_\perp := \mathcal{N} \cup \{\perp\}$, $\mathcal{R}_\perp := \mathcal{R} \cup \{\perp\}$, where $\perp$ denotes that an element is not observed. Then an *observation* is an element from $Obs := \mathcal{S}_\perp \times \mathcal{N}_\perp^3 \times \mathcal{R}_\perp$. For $\{S_0, S_1\} \subset \mathcal{S}$ and $\{R_0, R_1\} \subset \mathcal{R}$, we write $Obs(S_0, S_1, R_0, R_1) := \{S_0, S_1, \perp\} \times \mathcal{N}_\perp^3 \times \{R_0, R_1, \perp\}$ to denote the restriction of observations to these senders and recipients.

We abstractly model malicious infrastructure as the collection of all observation points of all Tor circuits that can be observed. This infrastructure comprises elements from $\mathcal{S} \times \mathcal{N}$ (connections between senders and nodes), elements from $\mathcal{N} \times \mathcal{N}$ (connection between Tor nodes), and elements from $\mathcal{N} \times \mathcal{R}$ (connections between nodes and recipients). Thus technically, a *malicious infrastructure* $\mathcal{MI}$ is an element of $\mathcal{I} := \mathcal{P}((\mathcal{S} \times \mathcal{N}) \cup (\mathcal{N} \times \mathcal{N}) \cup (\mathcal{N} \times \mathcal{R}))$ where $\mathcal{P}(X)$ denotes the powerset of set X.[2]

Underlying data structure. The probabilities of individual observations for a given sender-recipient

[1] As described in the last section, sender anonymity considers two senders S_0 and S_1 and one recipient R_1, and hence two such pairs (S_0, R_0) and (S_1, R_0). Recipient anonymity considers one sender S_0 and two recipients R_0 and R_1, and hence two such pairs (S_0, R_0) and (S_0, R_1). Relationship anonymity considers two senders S_0 and S_1 and two recipients R_0 and R_1, and hence four such pairs $(S_0, R_0), (S_0, R_1)$, (S_1, R_0) and (S_1, R_1).

[2] For node-level adversaries it is meaningful to consider adversaries that are only restricted by a budget, such as the number of nodes that they can compromise [14]. For malicious infrastructures such an unrestrictyed adversary is not meaningful, since it can, e.g., compromise a sender's AS and thereby win the sender anonymity game with probability 1.

```
ComputeAnonymity(MI, S_0, S_1, R_0, R_1)
  for each z ∈ {S_0, S_1} × {R_0, R_1} × Obs(S_0, S_1, R_0, R_1) do
    store[z] := 0
  store := ObservationPhase(store, MI, S_0, S_1, R_0, R_1)
  (δ_SA, δ_RA, δ_REL) := DeductionPhase(store, S_0, S_1, R_0, R_1)
  return (δ_SA, δ_RA, δ_REL)

ObservationPhase(store, MI, S_0, S_1, R_0, R_1)
  for each (s, r) ∈ {S_0, S_1} × {R_0, R_1}, (n_G, n_M, n_X) ∈ N^3 do
    store[s, r, obvd(MI, s, n_G, n_M, n_X, r)] += P_{s,r}(n_G, n_M, n_X)
  return store

obvd(MI, s, n_G, n_M, n_X, r)
  Initialize i := ⊥ for i ∈ {o_s, o_G, o_M, o_X, o_r}
  if (s, n_G) ∈ MI then o_s := s; o_G := n_G
  if (n_G, n_M) ∈ MI then o_G := n_G; o_M := n_M
  if (n_M, n_X) ∈ MI then o_M := n_M; o_X := n_X
  if (n_X, r) ∈ MI then o_X := n_X; o_r := d
  return (o_s, o_G, o_M, o_X, o_r)

DeductionPhase(store, S_0, S_1, R_0, R_1)
  δ_SA, δ_RA, δ_REL := 0
  for each o ∈ Obs(S_0, S_1, R_0, R_1) do
    AddDiff(δ_SA, store[S_0, R_0, o], store[S_1, R_0, o])
    AddDiff(δ_RA, store[S_0, R_0, o], store[S_0, R_1, o])
    r_1 := (store[S_0, R_0, o] + store[S_1, R_1, o])/2
    r_2 := (store[S_0, R_1, o] + store[S_1, R_0, o])/2
    AddDiff(δ_REL, r_1, r_2)
  return (δ_SA, δ_RA, δ_REL)

AddDiff(Z, X, Y)
  if X > Y then
    Z += X − Y
```

Figure 5.2: Computation of reduction of anonymity. Here $\mathcal{MI}$ denotes malicious infrastructure, S_0 and S_1 two senders, and R_0 and R_1 two recipients.

pair and a given malicious infrastructure are stored in a data structure `store`. `store` technically consists of an array of probabilities, addressed by elements of the form $(s, r, o) \in \mathcal{S} \times \mathcal{R} \times \mathit{Obs}$. All elements from `store` are initially set to 0.

Observation phase. For all four sender-recipient pairs (s, r), we first use the probability distribution $\mathrm{P}_{s,r}$, induced by Tor's current path selection algorithm or by LASTor, to determine the probability that a specific combination of guard node, middle node, and exit node is selected for (s, r). More formally, $\mathrm{P}_{s,r}$ in ObservationPhase constitutes a probability distribution over $\mathcal{N}^3$, where $\mathrm{P}_{s,r}(\mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X})$ denotes the probability that these three nodes are selected (in this order) to form a circuit from s to r.

Definition 1 (Path Selection Algorithms)**.** *A family of probabilistic algorithms* $\mathrm{P}_{s,r} \colon \mathcal{S} \times \mathcal{R} \to \mathcal{S} \times \mathcal{N}^3 \times \mathcal{R}$ *is a* path selection algorithm *if the following holds: for all* $S \in \mathcal{S}$ *and* $R \in \mathcal{R}$*, we have that if* $(S', n_1, n_2, n_3, R') \in [\mathrm{P}_{S,R}]$ *then* $S = S'$ *and* $R = R'$*, and* n_1, n_2, n_3 *are pairwise different.*

For each circuit from s to r we then derive all possible observations that can be successfully made by the considered malicious infrastructure $\mathcal{MI}$. To this end, we first determine all relevant sub-paths that constitute observation points for $\mathcal{MI}$, and then combine them to overall observation, i.e., to five-tuples from *Obs* that hence potentially include missing information ($\perp$). We stress that combining observations in this form (from individual sub-paths to five-tuples) corresponds to the adversary's ability to perfectly correlate Tor traffic – a worst-case assumption that reflects the

variety of successful works on Tor traffic correlation. For instance a malicious infrastructure that precisely observes the connection between a sender s and entry node n_G and between an exit node n_X and a recipient r corresponds to the observation $(s, \mathsf{n}_\mathsf{G}, \bot, \mathsf{n}_\mathsf{X}, r)$; hence the adversary is capable of linking s and r. Without traffic analysis, one would have to consider the tuples $(s, \mathsf{n}_\mathsf{G}, \bot, \bot, \bot)$ and $(\bot, \bot, \bot, \mathsf{n}_\mathsf{X}, r)$ individually, which we do not target here.

Finally, for each of these observations, we compute and store the aggregated probability over all Tor circuits that contain the respective Tor nodes.

Deduction phase. We now compute the adversary's advantage δ for the individual anonymity notions, based on the probabilities of the individual observations. To this end, we compare these probabilities and aggregate their differences in a suitable manner, depending on the considered anonymity notion:

- For *sender anonymity* δ_{SA}, we compare the probabilities of those observations that occur if either S_0 or S_1 are the sender of the circuit, with the same recipient R_0.
- For *recipient anonymity* δ_{RA}, we compare the probabilities of those observations that occur if S_0 is the sender of the circuit, and intends to establish a communication with either R_0 or R_1 as the recipient.
- For *relationship anonymity* δ_{REL}, we compare the probabilities of those observations for the two selected pairs arising from the four combinations of senders S_0 and S_1 and recipients R_0 and R_1, see Section 5.2.1.

5.3 Theoretical underpinning

In this section, we provide a rigorous semantics for the concepts that we informally used in the previous section, such as anonymity notions and the adversary's advantage. To this end, we cast all required formalizations in the AnoA framework [12], a recent framework for computing quantitative bounds for anonymous communication systems. Using this embedding into AnoA, we show that the bounds computed by algorithm ComputeAnonymity from Section 5.2 corresponds to the notion of adversary advantage in the AnoA framework, and thereby show that the bounds are accurate for Tor. Moreover, we will show that the bounds are tight under the assumption that traffic can be perfectly correlated.

We start by reviewing the definitions of the three anonymity notions in AnoA. After that, we show how to leverage the game-based definitions in AnoA for defining the adversary's advantage to the setting of malicious network infrastructure that we consider in this chapter.

5.3.1 Review: anonymity notions

The formalization of the three anonymity notions in AnoA closely follows the informal description that we gave in Section 5.2.1 as a challenge-response game in which the adversary has to distinguish two scenarios. Formally, an *anonymity notion* is simply a function α that receives as inputs two senders S_0 and S_1 and two recipients R_0 and R_1, as well as the so-called challenge bit b. It then selects one sender and one recipient, based on the challenge bit and the considered anonymity notion. For relationship anonymity, this selection is probabilistic. In slight abuse of notation we write S_b instead of "if $b = 0$ then S_0, else S_1", and similarly for R_b.

Sender anonymity α_{SA}. The sender anonymity function α_{SA} selects the sender according to the challenge bit, and always considers the same recipient R_0. Hence

$$\alpha_{SA}((S_0, S_1, R_0, R_1), b) := (S_b, R_0).$$

Recipient anonymity α_{RA}. The recipient anonymity function α_{RA} selects the recipient according to the challenge bit, and always considers the same sender S_0. Hence

$$\alpha_{RA}((S_0, S_1, R_0, R_1), b) := (S_0, R_b).$$

Relationship anonymity α_{REL}. The relationship anonymity function α_{REL} selects one of the four possible sender-recipient combinations as follows: if the challenge bit b is equal to 0, the function randomly selects one of the two pairs (S_0, R_0) or (S_1, R_1); if $b = 1$, it randomly selects between (S_0, R_1) and (S_1, R_0). Compactly, we obtain

$$\alpha_{RA}((S_0, S_1, R_0, R_1), b) := (S_{b'}, R_{b \oplus b'}); b' \leftarrow \{0, 1\}\,.$$

5.3.2 Leveraging game-based anonymity definitions

The definition of the AnoA challenger is the final part for casting the definition of the adversary's advantage in AnoA as a challenge-response game. In the AnoA framework, the challenger receives as input an anonymity notion α, two senders, two recipients, and the challenge bit, and it then simulates the Tor protocol for the sender-recipient scenario selected by α. The adversary interacts with the challenger, in order to determine which scenario is being simulated. The adversary knows all inputs to the challenger, in particular the anonymity notion, which senders and which recipients are being considered, which sender sends messages to which recipient, up to an uncertainty of one bit (the challenge challenge bit b).

In AnoA, one can furthermore provide to the challenger a description which parts of the infrastructure is considered malicious, and hence which observations an adversary is allowed to make in the interaction. As in Section 5.2 we characterize a malicious infrastructure as a set $\mathcal{MI} \subseteq \mathcal{P}((\mathcal{S} \times \mathcal{N}) \cup (\mathcal{N} \times \mathcal{N}) \cup (\mathcal{N} \times \mathcal{R}))$, where $\mathcal{P}(X)$ denotes the powerset of set X, and pass $\mathcal{MI}$ as an additional input to the challenger. We now describe the challenger in more detail.

The AnoA challenger for malicious infrastructure. The challenger `Ch` is defined in Figure 5.3. As described above, it expects as inputs the considered anonymity notion α, the challenge bit b, two senders S_0, S_1, two recipients R_0, R_1, and the description of the malicious infrastructure $\mathcal{MI}$. The challenger accepts two types of messages from the adversary: challenge-messages, denoted as $(\texttt{Challenge}, m)$ in Figure 5.3, for triggering that a dedicated challenge message is sent, and *input-messages*, denoted as $(\texttt{Input}, S, m, R)$ in Figure 5.3, for sending additional messages (m) between senders (S) and recipients (R):

- *Challenge-messages:* Upon receiving a message $(\texttt{Challenge}, m)$, the challenger computes the anonymity notion α on (S_0, S_1, R_0, R_1) and the challenge bit b, and obtains a sender-recipient pair $(S^*, R^*) \in \{S_0, S_1\} \times \{R_0, R_1\}$. The challenger then simulates the Tor protocol by creating a new Tor circuit $(\mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X})$ from sender S^* to recipient R^*, and then sends the message m from S^* to R^* using Tor. We abbreviate this using the subroutine SimulateTor(S^*, m, R^*) in Figure 5.3. Whenever a connection between two parties on this circuit can be observed according to $\mathcal{MI}$, i.e., whenever $(a, b) \in \mathcal{MI}$ for two consecutive elements a, b from the Tor circuit, the adversary is given the transcript of this communication. These observations precisely correspond to the observations described in Section 5.2.

- *Input-messages:* Upon receiving a message $(\texttt{Input}, S, m, R)$, the challenger calls the subroutine SimulateTor(S, m, R) as described in the previous case. Input-messages hence capture additional information the adversary may have about the communication contents in the Tor network.

We now define the *reduction of anonymity* for an anonymity function α as the adversary's advantage in this game.[3]

Definition 2. *(Reduction of anonymity/Advantage) Let α be an anonymity notion, S_0, S_1 two senders, R_0, R_1 two recipients, and $\mathcal{MI}$ a description of a malicious infrastructure. Then the* adversary's advantage *of an adversary $\mathcal{A}$ for these parameters is at most δ, with $0 \leq \delta \leq 1$, if for all sufficiently large $\eta \in \mathbb{N}$, we have*

$$\begin{aligned} &\Pr\left[0 = \langle \mathcal{A}(1^\eta) | \mathtt{Ch}(\alpha, 0, S_0, S_1, R_0, R_1, \mathcal{MI}) \rangle\right] \\ \leq\ &\Pr\left[0 = \langle \mathcal{A}(1^\eta) | \mathtt{Ch}(\alpha, 1, S_0, S_1, R_0, R_1, \mathcal{MI}) \rangle\right] + \delta. \end{aligned}$$

We say that Tor exhibits a reduction of anonymity *of at most δ, written as Tor is δ-IND-ANO, if the adversary's advantage of all probabilistic polynomial-time adversaries $\mathcal{A}$ is at most δ.*

This definition hence captures a static, passive adversary that controls a given subset of the network infrastructure. In particular, the adversary cannot adaptively decide which infrastructure parts to compromise.

5.3.3 Correctness of our algorithm

The algorithm ComputeAnonymity in Section 5.2 computes the probabilities of all observations that an adversary can make in a given scenario, and aggregates their differences in a scenario-dependent manner. We now show that the output of ComputeAnonymity corresponds to the anonymity reduction as defined in Definition 2, and hence indeed constitute meaningful anonymity bounds. Moreover, we show that these bounds are tight, under the assumption that perfect traffic correlation is possible.

We first show that the *optimal advantage* of an adversary can be characterized by the sum over the probability differences of the observations for all sub-paths, provided that we consider an abstraction of the Tor protocol. We call an advantage of δ *optimal* if the adversary's advantage is at most δ for all probabilistic polynomial-time adversaries $\mathcal{A}$, and if there exists an adversary $\mathcal{A}$ that achieves this advantage, i.e., the less-or-equal in Definition 2 is replaced by equality for $\mathcal{A}$. To define the abstraction of the Tor protocol, we define an abstract challenger $\mathtt{Ch}^*$. Instead of simulating the Tor protocol, $\mathtt{Ch}^*$ samples a Tor circuit $(\mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X})$, according to the probability distribution $\mathrm{P}_{S,R}$, and subsequently sends $\mathrm{OBVD}(\mathcal{MI}, S, \mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X}, R)$ to $\mathcal{A}$.

Definition 3 (Abstract Challenger)**.** $\mathtt{Ch}^*$ *is defined exactly as* $\mathtt{Ch}$ *in Figure 5.3 with the only difference that instead of the subroutine* SimulateTor $\mathtt{Ch}^*$ *uses* SimulateTor**, which is defined in Figure 5.4.*

This models that the adversary cannot gain information about the content of encrypted messages, but that it can still determine at which point in the challenge circuit it makes an observation (and still permitting traffic correlation attacks by definition of the challenger). The only exception is that if the observation is made between the exit node and the recipient, then the adversary would be able to see the message. However, since the adversary is allowed to choose the message in the interaction with the challenger anyway, observing it does not reveal additional information.

[3]The corresponding definition in AnoA additionally allows to consider a multiplicative advantage e^ε. We have set this to 1 here, such that δ directly corresponds to the reduction of anonymity, however, an analysis that captures the multiplicative advantage e^ε can be performed analogously. Moreover, AnoA considers arbitrary probabilistic, polynomial-time Turing machines and for technical reasons subsequently restricts them with wrapper machines (so-called adversary classes). For the sake of presentation in our specific setting, we did not introduce the lengthy description of adversary classes but instead restricted the adversary in the core definition and adjusted the challenger accordingly.

Challenger $\texttt{Ch}(\alpha, b, S_0, S_1, R_0, R_1, \mathcal{MI})$

Upon message $(\texttt{Input}, (S, m, R))$
Run SimulateTor(S, m, R).

Upon message $(\texttt{Challenge}, m)$
if this is the first challenge **then**
Compute $(S^*, R^*) \leftarrow \alpha((S_0, S_1, R_0, R_1), b)$
Run SimulateTor$(S^*, m, R^*, \mathcal{MI})$.
else
Abort the game.

Subroutine SimulateTor$(S, m, R, \mathcal{MI})$
Simulate the Tor protocol:
S builds a fresh Tor circuit C, yielding $(\mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X})$.
S sends m to R via the circuit C
for each two consecutive elements $x, y \in (S, \mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X}, R)$ **do**
if $(x, y) \in \mathcal{MI}$ **then**
Output the transcript sent on (x, y) to $\mathcal{A}$

Figure 5.3: The revised AnoA Challenger (for one challenge). Here α denotes the anonymity notion, b the challenge bit, S_0 and S_1 the two senders, R_0 and R_1 the two recipients, and $\mathcal{MI}$ the malicious infrastructure.

Subroutine SimulateTor$^*(S, m, R, \mathcal{MI})$
Sample a Tor circuit $(\mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X}) \leftarrow \mathrm{P}_{S,R}$
Let $o :=$ obvd$(\mathcal{MI}, S, \mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X}, R)$
Output o to $\mathcal{A}$.

Figure 5.4: SimulateTor*: The abstract instantiation of SimulateTor for the challenger $\texttt{Ch}^*$.

Lemma 1. *For every anonymity notion α_X with $X \in \{SA, RA, REL\}$, all senders S_0, S_1, all recipients R_0, R_1, and every description of a malicious infrastructure $\mathcal{MI}$, the optimal advantage of an adversary is equal to δ_X as computed by* ComputeAnonymity$(\mathcal{MI}, S_0, S_1, R_0, R_1)$ *from Section 5.2, if we assume an idealization of cryptography, and that traffic can be perfectly correlated.*

Proof. We prove the lemma in two steps. First, we show that the values for δ_X computed by ComputeAnonymity are upper bounds for the reduction of anonymity against the idealized challenger. Second, we show that an adversary exists that achieves this anonymity reduction, and hence that our bounds are tight.

Let α_X with $X \in \{SA, RA, REL\}$ be an anonymity notion, S_0, S_1 two senders, R_0, R_1, two recipients, and $\mathcal{MI}$ a description of a malicious infrastructure. To establish the upper bounds, we even consider an unbounded adversary $\mathcal{A}$. Under this assumption, $\mathcal{A}$ can pre-compute all computations of the challenger, given that cryptographic operations are removed in this abstract game. Hence input messages do not help the adversary in the game against the abstract challenger, since the simulated behavior on input messages is statistically independent of the behavior on challenge messages. We thus assume that $\mathcal{A}$ only receives a message from the challenger, which runs on a message $(\texttt{Challenge}, m)$ for an arbitrary but fixed m. For ease of notation, we omit the security parameter in the following, and let $(S^*, R^*) \leftarrow \alpha_X(S_0, S_1, R_0, R_1, b)$ denote the sender and recipient for the challenge. Moreover, we write Ω as a shortcut to denote the probability space on

the adversary's input induced by $\mathtt{Ch}^*(\alpha_X, 0, S_0, S_1, R_0, R_1, \mathcal{MI})$. We then obtain

$$\begin{aligned}
&\Pr\left[0 = \mathcal{A}(a) | a \leftarrow \mathtt{Ch}^*(\alpha_X, 0, S_0, S_1, R_0, R_1, \mathcal{MI})\right] \\
&= \Pr\left[0 = \mathcal{A}(a) | a \leftarrow \textsc{SimulateTor}^*(S^*, m, R^*, \mathcal{MI})\right] \\
&= \sum_{(\mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X})} \Big(\mathrm{P}_{S^*,R^*}(\mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X}) \\
&\qquad \cdot \Pr\left[0 = \mathcal{A}(\textsc{obvd}(\mathcal{MI}, S^*, \mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X}, R^*)) | \Omega\right] \Big) \\
&= \sum_{o \in Obs} \Bigg(\Big(\sum_{\substack{(\mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X}) s.t. \\ o = observe_{\mathcal{MI}}(S^*, \mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X}, R^*)}} \mathrm{P}_{S^*,R^*}(\mathsf{n_G}, \mathsf{n_M}, \mathsf{n_X}) \Big) \\
&\qquad \cdot \Pr\left[0 = \mathcal{A}(o) | \Omega\right] \Bigg) \\
&= \sum_{o \in Obs} \Big(\mathtt{store}[S^*, R^*, o] \cdot \Pr\left[0 = \mathcal{A}(o) | \Omega\right] \Big),
\end{aligned}$$

where $\mathtt{store}[S^*, R^*, o]$ is the probability that the observation o is made by the malicious infrastructure $\mathcal{MI}$, as calculated by the ComputeAnonymity algorithm after the ObservationPhase for the considered parameters $S_0, S_1, R_0, R_1, \mathcal{MI}$. The last equality holds by definition of the store, which contains the probability of each observation for a given pair of senders and recipients.

Upon inspection of Figure 5.2, depending on the anonymity notion, we conclude that ComputeAnonymity limits the success probability of $\mathcal{A}$ in the final DeductionPhase as follows. We exemplarily show the calculation for δ_{SA}; they follow analogously for δ_{RA} and δ_{REL}.

$$\begin{aligned}
&\sum_{o \in Obs} \Big(\mathtt{store}[S_0, R_0, o] \cdot \Pr\left[0 = \mathcal{A}(o) | \Omega\right] \Big) \\
&\leq \sum_{o \in Obs} \Big(\big(\mathtt{store}[S_1, R_0, o] + \\
&\qquad \big(\textbf{if } \mathtt{store}[S_0, R_0, o] > \big(\mathtt{store}[S_1, R_0, o] \textbf{ then} \\
&\qquad\quad \mathtt{store}[S_0, R_0, o] - \mathtt{store}[S_1, R_0, o] \textbf{ else } 0 \big) \\
&\qquad \cdot \Pr\left[0 = \mathcal{A}(o) | \Omega\right] \Big), \\
&\leq \sum_{o \in Obs} \Big(\mathtt{store}[S_1, R_0, o] \cdot \Pr\left[0 = \mathcal{A}(o) | \Omega\right] \Big) + \delta_{SA}.
\end{aligned}$$

This concludes the first part of the proof.

For the second part (bound tightness) we provide an adversary that achieves this bound. We again only show this for δ_{SA}; the remaining two cases work analogously. Consider an adversary $\mathcal{A}^*$ that computes for all possible observations $o \in Obs(S_0, S_1, R_0, R_1)$ the probabilities that they occur if $b = 0$ and if $b = 1$. Then $\mathcal{A}^*$ splits $Obs(S_0, S_1, R_0, R_1)$ into

$$\begin{aligned}
Obs_0 := \{ & o \in Obs(S_0, S_1, R_0, R_1) \\
& \mid \mathtt{store}[S_0, R_0, o] > \mathtt{store}[S_1, R_0, o] \}
\end{aligned}$$

and into $Obs_1 := Obs(S_0, S_1, R_0, R_1) \setminus Obs_0$. Whenever $\mathcal{A}^*$ observes any $o \in Obs_b$, it outputs b. The probability that $\mathcal{A}^*$ wins can be computed exactly by following the computation of δ_{SA} in Figure 5.2. □

Using Lemma 1 we can now show our main theorem.

Theorem 1. *For every anonymity notion α_X with $X \in \{SA, RA, REL\}$, all senders S_0, S_1 and recipients R_0, R_1, and every description of a malicious infrastructure $\mathcal{MI}$, the optimal advantage of a probabilistic polynomial-time adversary against Tor is equal to δ_X as computed by* ComputeAnonymity$(\mathcal{MI}, S_0, S_1, R_0, R_1)$ *from Section 5.2 up to a negligible additive factor, if we assume that invidual traffic observations can be perfectly correlated in probabilistic polynomial-time.*

Proof. The key idea of the proof is as follows: We first leverage previous work for showing that tight bounds are preserved for `Ch` with an idealized version of the Tor protocol. Subsequently we show that the tight bounds are preserved for the abstract challenger `Ch`* and finally apply Lemma 1 to prove the theroem.

The challenger `Ch` in Definition 2 can be represented as the challenger in the AnoA framework [12] with a suitable wrapper machine $AC_{\mathcal{MI},S_0,S_1,R_0,R_1}$ around the adversary $\mathcal{A}$ (a so-called adversary class in the AnoA framework), which depends on $\mathcal{MI}$, S_0, S_1, R_0, and R_1. We denote such a wrapped adversary as $AC_{\mathcal{MI},S_0,S_1,R_0,R_1}(\mathcal{A})$.

In a recent work [11], it has been shown that there is an idealized version of the Tor protocol that is securely realizable by the Tor protocol in the sense of the UC framework [16].

It has been shown in the AnoA framework that if an idealized protocol is δ-`IND-ANO` for some anonymity notion α_X and a certain adversary class, then there is a negligible function μ such that the cryptographic realization (in our case the Tor protocol) is $(\delta + \mu)$-`IND-ANO`.

It remains to be shown that for every adversary $AC_{\mathcal{MI},S_0,S_1,R_0,R_1}(\mathcal{A})$ against the idealized version of Tor there is an equally strong adversary against `Ch`* and, for the tightness, vice versa.

Since we assume perfect traffic correlation, the adversary is capable to determine if two individual observations for the same message sent over a circuit actually belong to the same circuit, and thereby connect both observations into a single one for that circuit. Thus, the main remaining hurdle in this proof is that the idealized Tor protocol from the literature does not reveal the position of an observation. The position, however, can be computed by an adversary by keeping track of the visible Tor nodes during the circuit construction phase as follows: due to the telescopic circuit construction, the number of messages sent for creating a circuit depends on the position of the entities; hence $\mathcal{A}$ can infer the position of the entity on the circuit by parsing the transcript and counting the number of times it observes for each entity.

For the tightness we use $\mathsf{n}_{\mathtt{dummy}}$ to describe protocol parties that the adversary cannot observe. Upon receiving an observation $o = (\mathsf{n}_1, \mathsf{n}_2, \mathsf{n}_3, \mathsf{n}_4, \mathsf{n}_5)$, the adversary replaces all unobserved elements $\mathsf{n}_i = \perp$ in o with $\mathsf{n}_{\mathtt{dummy}}$. Subsequently, we ask the sender n_1 (who might be $\mathsf{n}_{\mathtt{dummy}}$) to create a fresh Tor circuit consisting of the nodes $\mathsf{n}_2, \mathsf{n}_3$ and n_4 and we then send the message m over this circuit to n_5. Note that any of the Tor nodes involved and/or the recipient may be $\mathsf{n}_{\mathtt{dummy}}$.

Using Lemma 1, which shows that ComputeAnonymity$(\mathcal{MI}, S_0, S_1, R_0, R_1)$ computes the optimal adversary's advantage for the game against `Ch`*, we can conclude that $\delta + \mu$ is a bound for the anonymity reduction of the Tor protocol for some negligible function μ. Concerning tightness, the adversary defined in the second part of Lemma 1 indeed meets this bound δ_X.

□

5.3.4 Sessions and multiple challenges

The AnoA framework moreover allows to model sessions consisting of messages, i.e., challenges consisting of more than one message sent over the same Tor circuit). However, it has been shown that the usage of sessions does not impact Tor's anonymity [13]. It has also been shown there that

anonymity guarantees in single-challenge games can be transformed into anonymity guarantees for multiple challenges: If Tor is δ-IND-ANO-secure for one challenge, then it is $(n \cdot \delta)$-IND-ANO-secure for n challenges. This result trivially extends to the case of malicious network infrastructure.

Lemma 2. *If Tor is δ-IND-ANO for an anonymity notion α, two senders S_0, S_1, two recipients R_0, R_1, and a description of a malicious infrastructure $\mathcal{MI}$ (as in Definition 2), then Tor is also $(n \cdot \delta)$-IND-ANO for α, S_0, S_1, R_0, R_1, and $\mathcal{MI}$, for a modification of the challenger that accepts n messages $(\texttt{Challenge}, m)$ instead of aborting the game after the first such message.*

Proof. As in the proof of Theorem 1, this adjusted multiple-challenges challenger can be represented as the AnoA challenger [12] together with a suitable wrapper machine $AC_{\mathcal{MI},S_0,S_1,R_0,R_1}$ around the adversary, which depends on $\mathcal{MI}$, S_0, S_1, R_0, and R_1. Formally, our notion of δ-IND-ANO for the anonymity α in the sense of Definition 2 is defined in AnoA as $(1, 0, \delta, \alpha)$-IND-ANO with the adversary class $AC_{\mathcal{MI},S_0,S_1,R_0,R_1}$. Correspondingly, the extension of δ'-IND-ANO in this chapter to n challenges is defined in AnoA as $(n, 0, \delta', \alpha)$-IND-ANO with the adversary class $AC_{\mathcal{MI},S_0,S_1,R_0,R_1}$. Since this adversary class has a specific structure (called a Plug'n'Play Adversary Class in AnoA), it satisfies a property called single-challenge reducability. As shown in AnoA, for every adversary class that is single-challenge reducable, we have that $(1, 0, \delta, \alpha)$-IND-ANO implies $(n, 0, n \cdot \delta, \alpha)$-IND-ANO, which corresponds to the modification of the challenger to n-challenges.

□

5.4 Evaluation

In this section, we first describe how we constructed a model of the part of the Internet topology that is used by the Tor network, which we call *Tor's Internet topology*. Thereafter, we evaluate our derived model of Tor's Internet topology using the computation proposed in Section 5.2 to quantify the reduction of anonymity against several scenarios of malicious infrastructure. The scenarios include: *the ASes of several Tier-1 providers* (NTT, Level 3, DTAG) and their combination; the *German DE-CIX* as the largest Internet Exchange point worldwide; and the *landing point of Bude* that contains several submarine cables, one of them being the widely used TAT-14 cable between the US and Europe. We perform these computations both for Tor's path selection algorithm and for its variant LASTor [8], which was designed to reduce the impeding effects of malicious infrastructure by choosing geographically close Tor nodes. We first briefly review both path selection algorithms. We then describe how we implemented ComputeAnonymity from Section 5.2, how we selected senders and recipients for our analyses, and which malicious infrastructure we evaluated against. Finally, we present and discuss the corresponding results for the reduction of anonymity.

5.4.1 Tor's Internet topology

Large parts of the Internet topology are not publicly available. In particular, it might be involved to determine which routes a packet takes between two autonomous systems, which Internet exchange point it crosses or which submarine cables it traverses. A widely used source for constructing a model of the Internet's topology is publicly available BGP-data. Before we describe our model of Tor's Internet topology, we briefly discuss why a model purely based on BGP-data is too inaccurate for our purposes.

Inaccuracy of a purely BGP-based model

The Border Gateway Protocol (BGP) is the standard protocol for negotiating and announcing Internet routing between different ASes. In BGP, an AS announces to its peering ASes to which IP prefixes it can connect them, and these peers can relay this information to their respective peers. BGP data published by an AS consequently contains information about the paths that traffic will most likely take from exactly this AS to the announced IP prefixes. Since several organizations provide free BGP data feeds [7, 3], it is tempting to leverage BGP data for constructing a model of Internet routing. However, these local views on the Internet are each specific to one AS. To construct an accurate model of Tor's Internet topology, we would require BGP information from respective ASes of a large fraction of Tor nodes. Moreover, BGP information is very coarse-grained, as it only reveals the ASes on the route, but not the precise routes. Hence it lacks the geographical precision necessary for Internet exchange points and submarine cables, as many, especially the often-used Tier-1 ASes span dozens of countries and even several continents.

Our model of Tor's Internet topology

In order to evaluate our methodology for real-life settings, we constructed a model of Tor's Internet topology, i.e., how corresponding packets are routed in a snapshot of the Internet. Our topology is derived from 8.4 million unique routing paths from the iPlane service [29] that we suitably augmented with Traceroutes from Looking Glass servers [15]. This allows us to estimate the IP routes of communications between Tor nodes. Based on this topology, we can flexibly model a variety of network-level adversaries that control selected part of the network infrastructure, e.g., a subset of autonomous systems, and major communication hubs such as the world's largest Internet exchange point DE-CIX and the transatlantic submarine cable TAT-14. As an additional sanity check to refine precision and to exclude false positives, we use MaxMind's GeoIP2-City database [2] to take into account geographic proximity to landing points of submarine cables. We then empirically determined the coverage of our topology, i.e., the amount of routes between Tor relays that we determined, with respect to a best-effort model of today's Internet. Our findings show that our snapshot still consists of around 1650 Tor nodes for which we achieve a coverage of 97%.

5.4.2 Tor's default path selection and LASTor

In Section 5.2 we considered Tor's path selection algorithm as a given distribution over Tor circuits. For our evaluation, we concretely instantiate this distribution with a calculation of probabilities for all Tor circuits, depending on the (reduced) Tor network consensus. The dependence of Tor's path selection algorithm on a multitude of parameters (e.g., individual flags and weights of Tor nodes, family relations, TCP ports required for a connection, parameters selected by senders., etc.) makes this a non-trivial task.

In addition to Tor's default path selection algorithm, several variants have been proposed that strive to improve performance or anonymity under specific assumptions. Among these variants, LASTor [8] has been specifically designed to protect Tor's anonymity against malicious infrastructure. We hence include LASTor in our evaluation and compare its anonymity against the default selection algorithm. Thereby, we also provide answers to the question under which circumstances LASTor improves upon Tor's path selection algorithm, and identify situations where the opposite is the case.

Throughout our evaluation we compare senders that use the same variant of Tor's path selection algorithm (either the default one or LASTor), whereas the necessary TCP port(s) for the connection are determined by the respective recipients.

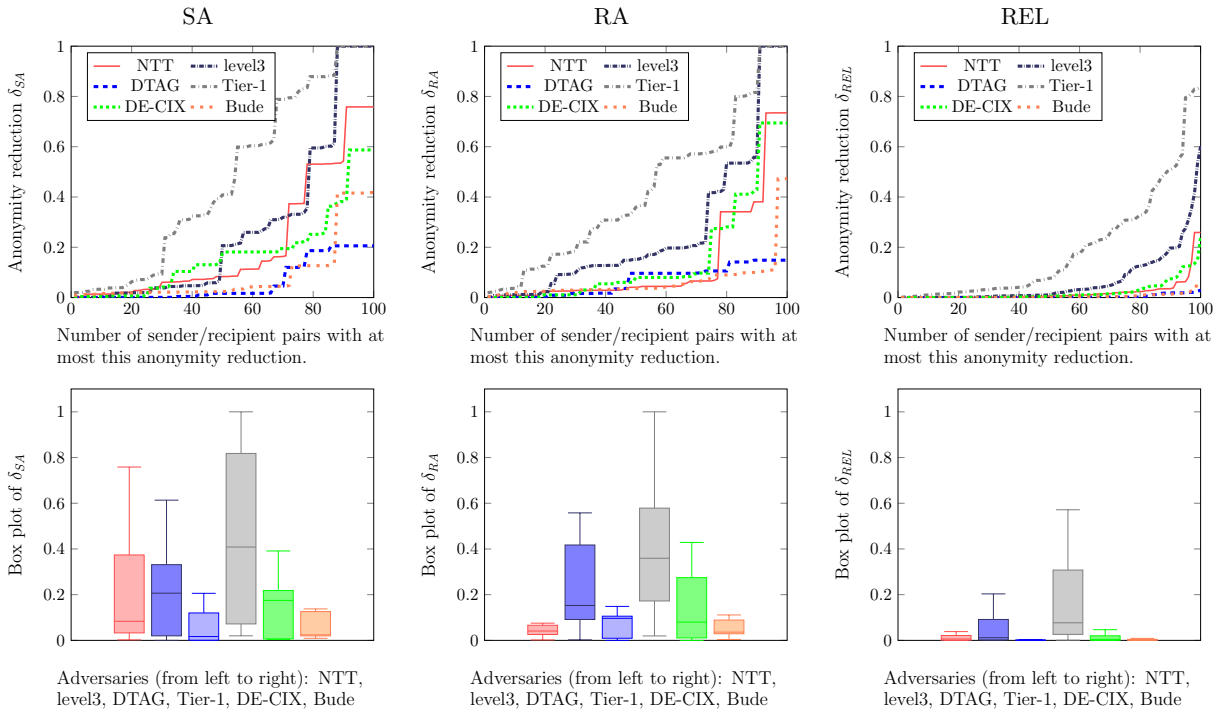

Figure 5.5: **PSTor:** Anonymity reduction per sample of senders and recipients (top), as well as box plots for these values, including lower quartile, median, upper quartile and minimal and maximal values of anonymity reduction (bottom) of Tor's path selection algorithm for all three anonymity notions: sender anonymity (left), recipient anonymity (middle) and relationship anonymity (right).

Tor's default path selection algorithm

For our evaluation we utilize MATor [13] for computing the distribution of Tor's path selection algorithm. Tor randomly selects nodes based on their flags in the Tor consensus, and for the exit node additionally based on whether the ports required by the user are offered by the Tor node. The path selection weights this random choice with the weight in the Tor consensus, which is highly correlated with the node's offered bandwidth. We refer to MATor [13] for a more detailed description.

LASTor

In LASTor, Tor nodes are grouped together into so-called clusters based on their physical location (latitude and longitude). LASTor first selects a guard cluster, a middle cluster and an exit cluster, and weights the guard cluster (and exit cluster) inverse to the distance between them and the sender (or the recipient, respectively), and thereby reduces the physical distance in expectation. After selecting clusters, LASTor selects a node from each cluster uniformly at random. To add further resilience against malicious ASes, LASTor ensures that no AS is chosen that can observe both the connections between sender and guard node, and between exit node and recipient. We refer to the original paper [8] for a more detailed description.

5.4.3 Senders and recipients

Routing path information from senders to Tor guards and vice versa, as well as from Tor exit nodes to recipients and vice versa, is crucial for accurately calculating the anonymity reduction against malicious infrastructure. Consequently, we select senders and recipients in a way that allows us to derive routing path information ourselves. To this end, we use looking glass servers – a collective of different servers, distributed over the world, that provide an open HTTP interface for running Traceroute. We use the server list provided at `http://www.traceroute.org/` as a reference for available looking glass servers, and we selected servers with a small response-time and high availability from this list. We then automatically query these servers with Traceroute request to all Tor guard nodes and Tor exit nodes via HTTP-requests and extract the Traceroute path from the resulting page. From all 290 working Looking Glass servers we utilized, we choose 20 stable and responsive servers with ideal coverage in different IP prefixes and select them as senders and recipients. We identified them using a tcpdump on a well-connected server under our control that captured any UDP, ICMP, and TCP traffic, and then queried Traceroute individually for our server from the used looking glass servers. We then identify the corresponding ASN via the whois service provided by team CYMRU and locate their geographical position via the commercial GeoIP database of MaxMind [2]. For each evaluation, we sampled two senders and two recipients from our 20 looking glass servers uniformly at random.

5.4.4 Evaluating malicious infrastructure

We now instantiate the malicious infrastructure of our analysis with actual companies and locations of interest. For ease of presentation and to achieve comparability of our results we focus on the following six instances of malicious infrastructure: a selection of three Tier-1 providers, as well as their combination; DE-CIX as the world's largest IXP; and one of the most important landing points of submarine cables at Bude, UK.

Nippon Telegraph and Telephone (NTT): All ASes belonging to the Japan-based Tier-1 provider NTT, which has more than 20 ASes, connecting Asia, North America and Europe.

Level 3 Communications: All 15 ASes belonging to the US-based Tier-1 provider Level 3 Communications, which operates in Africa, Asia, Europe, Middle East and South America.

Deutsche Telekom AG (DTAG): All 3 ASes belonging to the Germany-based Tier-1 provider DTAG, which operates in Africa, Asia, Europe, North America, and South America.

Tier-1: All ASes belonging to Level 3, DTAG and NTT.

DE-CIX: The world's largest IXP, the German DE-CIX in Frankfurt. The DE-CIX connects more than 626 ASes, with a strong focus on Europe. As a geographic constraint we only add DE-CIX to a routing path between IP prefixes of openly peering ASes within Europe.

Bude landing point: The landing point of submarine cables at Bude, UK. Hence the adversary can observe all communication that traverses this landing point. Several submarine cables are crossing through the Bude landing point: the transatlantic cables TAT-14, Apollo, and Yellow, the Euro-African-Indian cable Europe India Gateway, the France-Ireland-England cable FastnetConnect, the England-Westafrican cable Glo-1, and the England-Ireland cable Pan European Crossing.

5.4.5 Results

For our evaluation against the aforementioned scenarios of malicious infrastructure, we first consider Tor's default path selection algorithm and considered recipients that are only contacted via HTTPS

(TCP port 443). This corresponds to the main usage of Tor, as the Tor-Browser applies HTTPS-Everywhere. We thus consider senders and recipients for which the distribution over Tor circuits is equal, so that an adversary can gain information by observing traffic into the Tor network (for sender anonymity) or out of the Tor network (for recipient anonymity) or both (for relationship anonymity).

Our results for Tor's default path selection algorithm are given in Figure 5.5 (upper part) as three graphs, for sender anonymity, recipient anonymity, and relationship anonymity. In each graph, one line corresponds to one scenario of malicious infrastructure, showing all samples of sender/recipient pairs (x-axis) ordered by the respective anonymity reduction δ (y-axis) according to Definition 2. For ease of comparison, we added box plots for these graphs (Figure 5.5, lower part), presenting the lower quartile, the inner quartile (median) and the upper quartile, with whiskers showing the lowest and highest sample (excluding outliers with more than 1.5 times the difference between upper quartile and lower quartile, which rarely occurred).

We then compare our results for Tor's default path selection algorithm with our results for LASTor in Figure 5.6, depicting the difference between LASTors anonymity reduction and Tor's default anonymity reduction (per sample) for each scenario of malicious infrastructure.

Sender anonymity

For sender anonymity we considered a scenario where the sender visits a malicious website, i.e., connects to a malicious recipient. Consequently, the adversary can (in addition to the observations using malicious infrastructure) observe traffic from the Tor network to the recipient.

Tor: All considered adversaries noticeably reduce sender anonymity. Both Level 3 and NTT reduce sender anonymity significantly, for the majority of samples. The DE-CIX adversary is geographically restricted, and thus its success depends mainly on the location of the senders. In particular, DE-CIX reduces the anonymity of senders residing in Europe by more than 17% on average, while reducing anonymity by less than 1% for senders outside of Europe.

A corruption of the Bude landing point results in a reduction of sender anonymity ranging from 1% up to 40%, depending on the considered sample. We attribute these results to the fact that many Tor circuits need to use some submarine cable, as Tor circuits often span more than one continent. However, since we only considered a corruption of the Bude landing point, many other cables using different landing points exist.

LASTor: LASTor clearly improves anonymity against the DE-CIX adversary. However, the Level 3 adversary, and the Bude landing point adversary become significantly stronger. We interpret this as follows. The probability to choose a guard for which DE-CIX, or, for several samples, NTT or DTAG observes S_1-G is drastically reduced by LASTor. We see two possible reasons for the increased strength of the Level 3 adversary and the Bude adversary: first, the probability to use a guard to which they observe a communication from the sender is increased; and second, internal observations made within the Tor network give away information about the senders' location, e.g., in many cases a cable would be used by one sender, but not by another sender, even for traffic between the guard node and the middle node. Consequently, observations made within a Tor circuit can help in distinguishing between two senders. Although such an observation does not completely deanonymize the sender, it gives away partial information, and it substantiates the necessity to cover the more subtle information that an adversary can exploit. We attribute the overall increase of DTAG and NTT to the second reason as well.

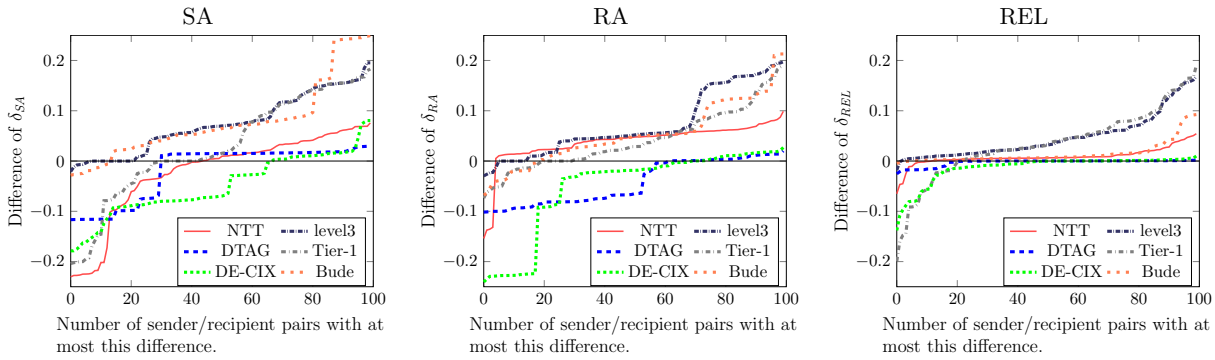

Figure 5.6: **Difference LASTor:** The difference between the anonymity reductions per sample of senders and recipients for all three anonymity notions: sender anonymity (left), recipient anonymity (middle) and relationship anonymity (right). Note that the y-axis ranges from -.25 to .25. A higher value means that LASTor offers in these cases less anonymity compared to the setting in Figure 5.5.

Recipient anonymity

For recipient anonymity we considered a scenario where the adversary can observe the Internet connection of the sender. Consequently, the adversary can (in addition to the observations using malicious infrastructure) observe traffic from the sender into the Tor network. As for sender anonymity, all considered adversaries noticeably reduce recipient anonymity. Level3 reduces anonymity more significantly than for sender anonymity: by over 10% for 70% of our samples. We observe that for all considered scenarios of malicious infrastructure, the variance of the anonymity reduction is significantly smaller for recipient anonymity than for sender anonymity. In particular, the anonymity reduction depends less on the positions of sender and recipient, and more on the positions of Tor's exit nodes. We attribute this dependency to the smaller number of Tor exit nodes.

Relationship anonymity

The relationship anonymity reduction of the considered malicious infrastructure is naturally smaller than for sender anonymity and for recipient anonymity, since relationship anonymity is inherently harder to break. Consequently, most considered scenarios of malicious infrastructures only reduced relationship anonymity by a small degree, with the noteworthy exception of the Tier-1 provider Level 3. Note that the combination of our three Tier-1 providers in comparison is extremely strong. For relationship anonymity, their combined impact is significantly higher than the sum of the impacts of the individual providers and reaches values above 20% for almost half of our samples. These results show that large (groups of) autonomous systems pose a significant threat for relationship anonymity, and thus, for every usage of Tor. Their impact is even stronger against LASTor, as even in cases in which these groups cannot completely deanonymize a connection, they can often gain partial information about the geographic location of senders and recipients.

5.4.6 Different TCP Ports

So far we analyzed Tor and LASTor for HTTPS only, which corresponds to the main usage of Tor. However, Tor can be used for (almost) arbitrary TCP traffic. For the following analyses, we hence compare two recipients with different requirements: one is only contacted via HTTPS on port 443 and the other provides an Internet Relay Chat (IRC), thereby additionally requiring port 6667.

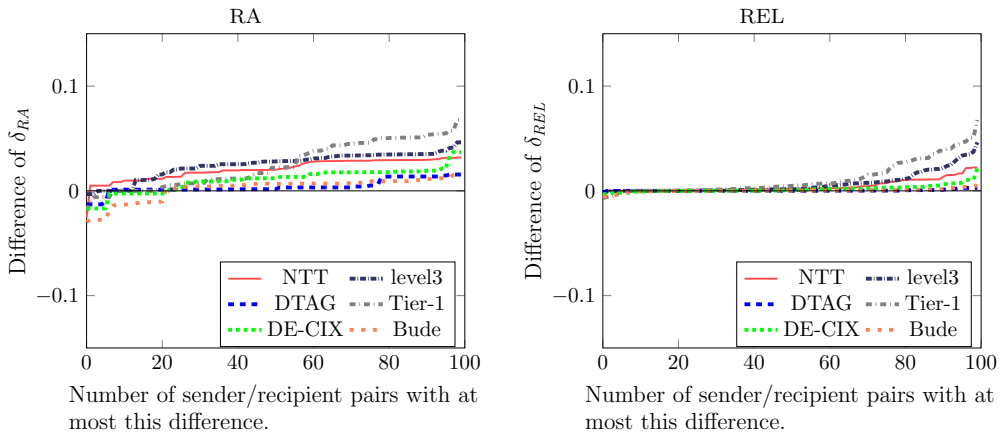

Figure 5.7: **Difference Distinct Ports PSTor:** Impact of using different TCP ports on the anonymity reduction per sample of senders and recipients of Tor's path selection algorithm for recipient anonymity (left) and relationship anonymity (right). Note that the y-axis ranges from -0.15 to 0.15. We compare a user requesting port 443 (HTTPS) with a user requesting port 443 and 6667 (IRC). A higher value means that in these cases different ports with PSTor leads to less anonymity compared to the setting in Figure 5.5.

As this port is not supported by all Tor exit nodes, the path selection algorithm is restricted. We present the anonymity reduction imposed by the considered scenarios of malicious infrastructure in Figure 5.7 (for Tor's path selection algorithm). Considering ports typically increases the anonymity reduction. However, in contrast to the observations of Backes et. al. [13] for compromised Tor nodes (where restricting the path selection algorithm for $S_0{-}R_1$, but not for $S_0{-}R_0$, always reduces anonymity), we can identify cases in which the anonymity reduction is less severe if one recipient requires a less supported port, like IRC (see, e.g., the leftmost results for Bude in Figure 5.7). For instance, consider a heavy-weight exit node X that only supports port 443 and an adversary that observes the connection between X and R_1, but not between X and R_0. While the adversary can deanonymize the recipient if it observes traffic from X to R_1, no such observation can be made anymore if R_1 requires a different port, since X cannot be selected as an exit node.

5.5 Related work

There is a rich literature on traffic correlation attacks against Tor [32, 23, 21, 30, 22, 10, 34]. These works substantiate the common belief that malicious infrastructure adversaries with their traffic correlation attacks constitute the most important real-life threat to Tor. We discuss the most closely related works in this respect in the following. None of these works aimed at providing semantically well-founded bounds to which extent anonymity is being reduced if specific parts of the network infrastructure are considered malicious; we do not repeat this for all the following works.

Previous work on establishing rigorous anonymity guarantees (e.g., [20], AnoA [12], and MA-Tor [13]) only consider node-level adversaries and do not address network-level adversaries, such as Tier-1 providers, Internet Exchange points, or submarine cables. However, we ground the algorithmic computation conducted here and its theoretical underpinning on the AnoA framework, in order to for quantifying the degree of a user's anonymity in an anonymous communication service,

such as Tor [17], Aqua [28], or I2P [24].

Nithyanand et al. [33] showed how to empirically measure the impact of AS-level adversaries against Tor. They ground their work on AS-level peering data (including customer, peer, and provider relations) and use shortest AS paths and randomization in case several options are available. As such, it provides empirical observations without striving for rigorous guarantees. In contrast to our derived model, their model consists of AS paths instead of IP prefixes, and it in particular disregards that large ASes would, depending on the destination of the source, route a packet in a completely different way. A recent study by Anwar et al. [9] shows that ignoring the geographic position of source and destination can lead to up to 45% of inaccuracy. In particular on intercontinental routing paths, the AS-level model is highly inaccurate. In the presence of Tor's path selection that (for anonymity reasons) ignores a client's geographic position and thereby chooses with high probability paths that include transcontinental sub-paths, the shortcomings of such a model become evident.

Juen et al. [27] analyzed the precision of routing path prediction techniques for anonymity analysis of Tor and improved path selection algorithms. The authors presented a method for Tor relays to measure large parts of the Internet structure, and succeeded in convincing 28 Tor relays to deploy their method. Additionally, they briefly analyzed Tor against singular network adversaries (the same AS twice on a path). It would be interesting to use their tools for gathering data about the Internet's topology to complement our model, and potentially vice versa.

Jaggard et al. [25] also developed a trust-based analysis framework for which they also consider malicious submarine cables. Their work is complementary to ours, since their focus lies on formalizing the trust on a given piece of infrastructure.

Johnson et al. [26] introduced a simulation framework for Tor's path selection and analyzed the benefit of guard nodes and the strength of AS-level adversaries. The underlying model of the Internet topology they use, however, considers not only AS-level paths, which often strongly reduce accuracy (see above), but also randomly chosen paths through neighboring ASes. Their work is insightful, but the resulting imprecision of their model of the Internet topology distorts the results and makes them hard to interpret.

These network-level attacks gave rise to several novel path selection algorithms that reduce the number of ASes that are crossed, such as LASTor by Akhoondi et al. [8] or the AS-aware Tor path selection by Edman and Syverson [18]. In this work, we compare the anonymity guarantees of LASTor (in our model of the Internet's topology) with Tor's path selection, and thereby quantitatively answer the question if minimizing the number of crossed ASes increases or decreases a client's anonymity. As our evaluation shows, both answers are possible, depending on the location of senders and recipients and on the position of the AS: in many cases a client's anonymity can indeed be increased; yet, the necessary methods inherently leak partial information about clients, and hence present an additional attack vector for malicious ASes.

Acknowledgements. This project was performed in collaboration with Esfandiar Mohammadi, Simon Koch, Christian Rossow and Michael Backes.

Bibliography

[1] Collector – your friendly data-collecting service in the tor network. `https://collector.torproject.org/`, accessed 2015.

[2] Maxind geoip2-city database. `https://www.maxmind.com/de/geoip2-databases`, accessed 2015.

[3] Ripe ris raw data. `https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data`, accessed 2015.

[4] Tails: the amnestic incognito live system. `https://tails.boum.org/`, accessed 2015.

[5] Tor metrics. `https://metrics.torproject.org/`, accessed 2015.

[6] Torproject. `https://torproject.org`, accessed 2015.

[7] University of oregon route views project. `http://www.routeviews.org/`, accessed 2015.

[8] M. Akhoondi, C. Yu, and H. V. Madhyastha. Lastor: A low-latency as-aware tor client. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 476–490. IEEE, 2012.

[9] R. Anwar, H. Niaz, D. Choffnes, I. Cunha, P. Gill, and E. Katz-Bassett. Investigating interdomain routing policies in the wild. Technical report, University of Southern California, 2015.

[10] D. Arp, F. Yamaguchi, and K. Rieck. Torben: A practical side-channel attack for deanonymizing tor communication. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 597–602. ACM, 2015.

[11] M. Backes, A. Kate, I. Goldberg, and E. Mohammadi. Provably Secure and Practical Onion Routing. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium (CSF)*, pages 369–385. IEEE, 2012.

[12] M. Backes, A. Kate, P. Manoharan, S. Meiser, and E. Mohammadi. AnoA: A Framework For Analyzing Anonymous Communication Protocols. Cryptology ePrint Archive, Report 2014/087, 2015.

[13] M. Backes, A. Kate, S. Meiser, and E. Mohammadi. (Nothing else) MATor(s): Monitoring the Anonymity of Tor's Path Selection. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, pages 513–524. ACM, 2014.

[14] M. Backes, S. Meiser, and M. Slowik. Your choice mator (s). *Proceedings on Privacy Enhancing Technologies*, 2016(2):40–60, 2015.

[15] BGP4AS. Bgp looking glasses for ipv4/ipv6, traceroute & bgp route servers. Accessed 2015, available at: `http://www.bgp4.as/looking-glasses`.

[16] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2013.

[17] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[18] M. Edman and P. Syverson. As-awareness in tor path selection. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 380–389. ACM, 2009.

[19] N. Feamster and R. Dinglediner. Location diversity in anonymity networks. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, pages 66–76. ACM, 2004.

[20] J. Feigenbaum, A. Johnson, and P. Syverson. Probabilistic analysis of onion routing in a black-box model. *ACM Trans. Inf. Syst. Secur.*, 15(3):14:1–14:28, 2012.

[21] A. Houmansadr and N. Borisov. Swirl: A scalable watermark to detect correlated network flows. In *In Proceedings of the 18th Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2011.

[22] A. Houmansadr and N. Borisov. The need for flow fingerprints to link correlated network flows. In *Proceedings of the 13th International Symposium on Privacy Enhancing Technologies (PETS)*, volume 7981 of *LNCS*, pages 205–224. Springer Berlin Heidelberg, 2013.

[23] A. Houmansadr, N. Kiyavash, and N. Borisov. RAINBOW: A robust and invisible non-blind watermark for network flows. In *Proceedings of the 17th Network and Distributed System Security Symposium (NDSS)*, 2009.

[24] I2P. The invisible internet project. Accessed 2015, available at: `https://geti2p.net`.

[25] A. D. Jaggard, A. Johnson, S. Cortes, P. Syverson, and J. Feigenbaum. 20,000 in league under the sea: Anonymous communication, trust, mlats, and undersea cables. In *Proceedings of the 15th Privacy Enhancing Technologies Symposium (PETS)*, pages 4–24. De Gruyter Open, 2015.

[26] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users get routed: Traffic correlation on tor by realistic adversaries. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS)*, pages 337–348. ACM, 2013.

[27] J. Juen, A. Das, A. Johnson, N. Borisov, and M. Caesar. Defending tor from network adversaries: A case study of network path prediction. *CoRR*, abs/1410.1823, 2014.

[28] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis. Towards efficient traffic-analysis resistant anonymity networks. *SIGCOMM Comput. Commun. Rev.*, 43(4):303–314, 2013.

[29] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane: An information plane for distributed services. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 367–380. USENIX Association, 2006.

[30] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, pages 215–226. ACM, 2011.

[31] Mozilla. Introducing polaris privacy initiative to accelerate user-focused privacy online. `https://blog.mozilla.org/privacy/2014/11/10/introducing-polaris-privacy-initiative-to-accelerate-user-focused-privacy-online/`, accessed 2015.

[32] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *Security and Privacy, 2005 IEEE Symposium on*, pages 183–195. IEEE, 2005.

[33] R. Nithyanand, O. Starov, A. Zair, P. Gill, and M. Schapira. Measuring and mitigating as-level adversaries against tor. eprint arXiv:1505.05173, 2015.

[34] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. Raptor: Routing attacks on privacy in tor. *arXiv preprint arXiv:1503.03940*, 2015.

[35] L. Vanbever, O. Li, J. Rexford, and P. Mittal. Anonymity on quicksand: Using bgp to compromise tor. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 14. ACM, 2014.

A. The SphinxMix Python package

The `sphinxmix` python package is an open source library, published by PANORAMIX partners under a LPGL license, that implements the core cryptographic functions necessary for building a decryption mix network. The cryptographic techniques are based on the `sphinxmix` packet format (Danezis & Goldberg 2009), and have been significantly extended for performance and flexibility. In particular, the library allows for arbitrary replay prevention mechanisms to be implemented, for arbitrary routing information to be provided to each mix on a message path, and for the ease use of any cryptographic primitives for public key and symmetric key cryptographic operations. This flexibility is necessary to experiment as part of WP3 with different research prototypes, but the code base is of production quality – and the library is available for all to use.

The `sphinxmix` package master branch is about 700 lines of cryptographic code, including the functional components, documentation, tests and configuration. A development branch, of similar size and complexity, is also available testing lower level C language bindings for key cryptographic operations using the `Cython` compiler. Where performance improvements are significant, features are being integrated in the master branch.

The `sphinxmix` development follows gold standards in terms of open source engineering quality and tool chains:

Revision control & issue tracking. The library is developed under a regime of distributed revision control, using the `git` revision control systems. Development is done on a number of temporary feature branches and merged into a master release branch after testing. It is actively maintained and hosted on the GitHub service and is available for all to clone, report issues and contribute patches to.

Unit tests & CI. The library contains tests for all functions implemented using the well-established `pytest` framework. Testing for compatibility with Python 2 and Python 3 is automated using the `Tox` framework, that also tests its packaging logic. The on-line continuous integration service Travis-CI is used to monitor all branches and ensure that all tests pass on all platforms. Test coverage is 97% of the lines of code and is continuously monitored for each build by the `coveralls.io` service.

Packaging & availability. The library is available as a standalone Python package, along with a clear description of its dependencies that can be automatically installed using the standard `pip` package manager. Current and all previous versions of the library are available from the central Python package repository `Pypi`.

Online & self documentation. All user-exposed library functions are self-documented using `pydoc` strings, and thematic example usages and a guide is also provided for the package. The documentation examples are tested as part of the test suite, and the documentation complied using the `Sphinx` Python documentation package. The documentation is available on the standard `pythonhosted.org` service.

This appendix presents the library documentation and user guide.

The sphinxmix package documentation

This documentation relates to the *sphinxmix* package version 0.0.6.

Installing

Install using *pip* through the command:

```
$ pip install sphinxmix
```

Basic usage

The `sphinxmix` package implements the Sphinx mix packet format core cryptographic functions.

The paper describing sphinx may be found here:

- George Danezis and Ian Goldberg. Sphinx: A Compact and Provably Secure Mix Format. IEEE Symposium on Security and Privacy 2009. [link]

All the `sphinxmix` cryptography is encapsulated and within a `SphinxParams` object that is used by all subsequent functions. To make `sphinxmix` use different cryptographic primitives simply extend this class, or re-implement it. The default cryptographic primitives are `NIST/SEGS-p224` curves, `AES` and `SHA256`.

Sending Sphinx messages

To package or process sphinx messages create a new `SphinxParams` object:

```
>>> # Instantiate a the crypto parameters for Sphinx.
>>> from sphinxmix.SphinxParams import SphinxParams
>>> params = SphinxParams()
```

The `sphinxmix` package requires some rudimentary Public Key Information: mix nodes need an identifier created by `Nenc` and the PKI consists of a dictionary mapping node names to `pki_entry` records. Those include secret keys (derived using `gensecret`) and public keys (derived using `expon`).

```
>>> # The minimal PKI involves names of nodes and keys
>>> from sphinxmix.SphinxClient import pki_entry, Nenc
>>> pkiPriv = {}
>>> pkiPub = {}
>>> for i in range(10):
...     nid = i
...     x = params.group.gensecret()
...     y = params.group.expon(params.group.g, x)
...     pkiPriv[nid] = pki_entry(nid, x, y)
...     pkiPub[nid] = pki_entry(nid, None, y)
```

A client may package a message using the Sphinx format to relay over a number of mix servers. The function `rand_subset` may be used to select a random number of node identifiers; the function `create_forward_message` can then be used to package the message, ready to be sent to the first mix. Note both destination and message need to be `bytes`.

```
>>> # The simplest path selection algorithm and message packaging
>>> from sphinxmix.SphinxClient import rand_subset, \
...                                    create_forward_message
>>> use_nodes = rand_subset(pkiPub.keys(), 5)
>>> nodes_routing = list(map(Nenc, use_nodes))
>>> keys_nodes = [pkiPub[n].y for n in use_nodes]
>>> dest = b"bob"
>>> message = b"this is a test"
>>> header, delta = create_forward_message(params, nodes_routing, \
...     keys_nodes, dest, message)
```

The client may specify any information in the `nodes_routing` list, that will be passed to intermediate mixes. At a minimum this should include information about the next mix.

Processing Sphinx messages at a mix

The heart of a Sphinx mix server is the `sphinx_process` function, that takes the server secret and decodes incoming messages. In this example the message encode above, is decoded by the sequence of mixes.

```
>>> # Process message by the sequence of mixes
>>> from sphinxmix.SphinxClient import PFdecode, Relay_flag, Dest_flag, Surb_flag, receive_forward
>>> from sphinxmix.SphinxNode import sphinx_process
>>> x = pkiPriv[use_nodes[0]].x
>>> while True:
...     ret = sphinx_process(params, x, header, delta)
...     (tag, info, (header, delta)) = ret
...     routing = PFdecode(params, info)
...     if routing[0] == Relay_flag:
...         flag, addr = routing
...         x = pkiPriv[addr].x
...     elif routing[0] == Dest_flag:
...         assert receive_forward(params, delta) == [dest, message]
...         break
```

It is the responsibility of a mix to record `tags` of messages to prevent replay attacks. The `PFdecode` function may be used to recover routing Information including the next mix, or any other user specified information.

Single use reply Blocks

A facility provided by Sphinx is the creation and use of Single Use Reply Blocks (SURB) to route messages back to an anonymous receipient. First a receiver needs to create a SURB using `create_surb` and passes on the `nymtuple` structure to the sender, and storing `surbkeytuple` keyed by the identifier `surbid`:

```
>>> from sphinxmix.SphinxClient import create_surb, package_surb
>>> surbid, surbkeytuple, nymtuple = create_surb(params, nodes_routing, keys_nodes, b"myself")
```

Using the `nymtuple` a sender can package a message to be sent through the network, starting at the `nymtuple[0]` router:

```
>>> message = b"This is a reply"
>>> header, delta = package_surb(params, nymtuple, message)
```

The network processes the SURB as any other message, until it is received by the last mix in the path:

```
>>> x = pkiPriv[use_nodes[0]].x
>>> while True:
...    ret = sphinx_process(params, x, header, delta)
...    (tag, B, (header, delta)) = ret
...    routing = PFdecode(params, B)
...
...    if routing[0] == Relay_flag:
...        flag, addr = routing
...        x = pkiPriv[addr].x
...    elif routing[0] == Surb_flag:
...        flag, dest, myid = routing
...        break
```

The final mix server must sent the `myid` and `delta` to the destination `dest`, where it may be decoded using the `surbkeytuple`.

```
>>> from sphinxmix.SphinxClient import receive_surb
>>> received = receive_surb(params, surbkeytuple, delta)
>>> assert received == message
```


Embedding arbitrary information for mixes

A sender may embed arbitrary information to mix nodes, as demonstrated by embedding `b'info'` to each mix, and `b'final_info'` to the final mix:

```
>>> use_nodes = rand_subset(pkiPub.keys(), 5)
>>> nodes_routing = [Nenc((n, b'info')) for n in use_nodes]
>>> keys_nodes = [pkiPub[n].y for n in use_nodes]
>>> dest = (b"bob", b"final_info")
>>> message = b"this is a test"
>>> header, delta = create_forward_message(params, nodes_routing, \
...     keys_nodes, dest, message)
```

Mixes decode the arbitrary structure passed by the clients, and can interpret it to implement more complex mixing strategies:

```
>>> x = pkiPriv[use_nodes[0]].x
>>> while True:
...     ret = sphinx_process(params, x, header, delta)
...     (tag, info, (header, delta)) = ret
...     routing = PFdecode(params, info)
...     if routing[0] == Relay_flag:
...         flag, (addr, additional_info) = routing
...         assert additional_info == b'info'
...         x = pkiPriv[addr].x
...     elif routing[0] == Dest_flag:
...         [[dest, additional_info], msg] = receive_forward(params, delta)
...         assert additional_info == b'final_info'
...         assert dest == b'bob'
...         break
```

Packaging mix messages to byte strings:

The *sphinxmix* package provides functions *pack_message* and *unpack_message* to serialize and deserialize mix messages using *msgpack*. Some meta-data about the parameter length are passed along the message any may be used to select an appropriate parameter environment for the decoding of the message.

```
>>> from sphinxmix.SphinxClient import pack_message, unpack_message
>>> bin_message = pack_message(params, (header, delta))
>>> param_dict = { (params.max_len, params.m):params }
>>> px, (header1, delta1) = unpack_message(param_dict, bin_message)
```

Development

The git repository for `sphinxmix` can be cloned from here: https://github.com/UCL-InfoSec/sphinx

The `pytest` unit tests and doctests of `sphinxmix` may be ran using `tox` simply through the command:

```
$ tox
```

To upload a new distribution of `sphinxmix` the maintainer simply uses:

```
$ python setup.py sdist upload
```

Core classes and functions

class `sphinxmix.SphinxParams.SphinxParams`(*group=None*, *header_len=192*, *body_len=1024*)

class `sphinxmix.SphinxParams.Group_ECC`(*gid=713*)

Group operations in ECC

Client functions

`sphinxmix.SphinxClient.pki_entry`(*id*, *x*, *y*)

A helper named tuple to store PKI information.

sphinxmix.SphinxClient.**Nenc**(*idnum*)

The encoding of mix names.

sphinxmix.SphinxClient.**Relay_flag** = *'\xf0'*

Routing flag indicating message is to be relayed.

sphinxmix.SphinxClient.**Dest_flag** = *'\xf1'*

Routing flag indicating message is to be delivered.

sphinxmix.SphinxClient.**Surb_flag** = *'\xf2'*

Routing flag indicating surb reply is to be delivered.

sphinxmix.SphinxClient.**PFdecode**(*param*, *packed*)

Decoder of prefix free encoder for commands received by mix or clients.

sphinxmix.SphinxClient.**rand_subset**(*lst*, *nu*)

Return a list of nu random elements of the given list (without replacement).

sphinxmix.SphinxClient.**create_forward_message**(*params*, *nodelist*, *keys*, *dest*, *msg*)

Creates a forward Sphix message, ready to be processed by a first mix.

It takes as parameters a node list of mix information, that will be provided to each mix, forming the path of the message; a list of public keys of all intermediate mixes; a destination and a message (byte arrays).

sphinxmix.SphinxClient.**receive_forward**(*params*, *delta*)

Decodes the body of a forward message.

Packaging messages

sphinxmix.SphinxClient.**pack_message**(*params*, *m*)

A method to pack mix messages.

sphinxmix.SphinxClient.**unpack_message**(*params_dict*, *m*)

A method to unpack mix messages.

Mix functions

sphinxmix.SphinxNode.**sphinx_process**(*params*, *secret*, *header*, *delta*)

The heart of a Sphinx server, that processes incoming messages. It takes a set of parameters, the secret of the server, and an incoming message header and body.

SURB functions

sphinxmix.SphinxClient.**create_surb**(*params*, *nodelist*, *keys*, *dest*)

Creates a Sphinx single use reply block (SURB) using a set of parameters; a sequence of mix identifiers; a pki mapping names of mixes to keys; and a final destination.

Returns:

- A triplet (surbid, surbkeytuple, nymtuple). Where the surbid can be used as an index to store the secrets surbkeytuple; nymtuple is the actual SURB that needs to be sent to the receiver.

sphinxmix.SphinxClient.**package_surb**(*params*, *nymtuple*, *message*)

Packages a message to be sent with a SURB. The message has to be bytes, and the nymtuple is the structure returned by the create_surb call.

Returns a header and a body to pass to the first mix.

sphinxmix.SphinxClient.**receive_surb**(*params*, *keytuple*, *delta*)

Processes a SURB body to extract the reply. The keytuple was provided at the time of SURB creation, and can be the SURB id, which is also returned to the receiving user.

Returns the decoded message.

Indices and tables

- Index
- Module Index
- Search Page