Panos Louridas (GRNET)
George Tsoukalas (GRNET)
Dimitris Mitropoulos (GRNET)

Integrated Service

Deliverable D5.3

April 30, 2018
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	2018-20-03	DM, GT, PL (GRNET)	Proposed table of contents
0.2	2018-30-03	DM, GT (GRNET)	First draft of Integration section
0.3	2018-05-04	DM, GT (GRNET)	Architecture added
0.4	2018-13-04	DM, GT (GRNET)	Added Requirements
0.5	2018-20-04	DM, GT (GRNET)	Added Validation
0.6	2018-23-04	DM, GT (GRNET)	Added Introduction and Conclusions
0.7	2018-24-04	DM, GT (GRNET)	Added Summary
0.8	2018-28-04	TZ (UEDIN)	Review Comments
0.81	2018-28-04	PC (UOA)	Review Comments
0.9	2018-29-04	DM, GT (GRNET)	Addressed review comments
1.0	2018-30-04	MW (UEDIN)	Final version submitted to EC

Executive Summary

This deliverable describes an updated version of the Zeus e-voting system, which is being developed in the context of Work Package 5. It is actually an integrated service which means that Zeus is running in the context of PANORAMIX and can work with different mix-nets to carry out an electronic election. In essence, the integrated service implements the full feature set and incorporates adjustments after the experience with the PANORAMIX MVP (described in Deliverable 5.2).

In Section 2, we enumerate a number of basic requirements that were unfulfilled in the MVP, and have been fulfilled by the current version of the integrated service. Then, in Section 3, we present how Zeus was fully ported to PANORAMIX (note that the system was described in Deliverable 4.3). Furthermore, to validate the integrated service, we have integrated two well-known mix-nets: Bayer Groth's and and a re-encryption mix-net developed in the context of the PANORAMIX project, to enhance Zeus' efficiency and effectiveness. We discuss these advances in Section 4.

Contents

1. Introduction

The purpose of this document is to present the updated version of Zeus [12], an e-voting application developed by GRNET. Zeus has been in production since late 2012. So far, it has been used in more than 520 elections for state and private organizations. Zeus is the centerpiece of Work Package 5 which involves the first PANORAMIX (Privacy and Accountability in Networks via Optimized Randomized Mix-nets) use-case. Through our work in WP5 we have been able to raise e-voting performance by integrating more than one mix-net into the updated version of Zeus.

This deliverable describes the Zeus integrated system, which implements the remaining requirements which were not addressed in the version presented in Deliverable 5.2 (Section 2). It constitutes a complete system with a stable Application Programming Interface (API), and a solid architecture based on the integrated system presented in Deliverable 4.3 (Section 3). The architecture of the system lets us integrate two mix-nets and validate our implementation successfully (Section 4).

We need to note here that in the context of PANORAMIX and in the e-voting case in particular, we wanted to encourage mix-net technology to be adopted in an environment where responsibility for user privacy is being legislated as a core component of the information technology industry. For instance, the European Union General Data Protection Regulation (GDPR) [1] replaces central certification authorities with self-assessment processes. This shifts the weight of risk analysis and security responsibility to the individual organization. Our service was designed for increased transparency and control of technical parameters and as such it will be a valuable asset for meeting the regulatory requirements.

2. Fulfilled requirements for the e-voting use-case

In this section we will present a number of key requirements that we have satisfied via the integrated product These requirements are quite important in the e-voting context, because trustees must be able to control the election procedure, including mixing. This requires a usable and intuitive user interface that will present all decisions and actions in a uniform way so that the trustees can have sufficient overview themselves and not rely on delegation to more technically trained operators. Usability will also make finer control and overview possible. An important part of control is the audit log of all actions so that trustees can both be held accountable for the procedure and prove to auditors that they observed due diligence.

There are also several general requirements fulfilled by the current version of the integrated service. Note that these requirements were not satisfied by the MVP (see Deliverable 5.2). We discuss these requirements in detail. Note that a more generic description of such requirements was discussed in Deliverable 4.3.

2.1 Peer Authentication

To authenticate the different entities that exist in the e-voting context and participate in the corresponding protocols, the integrated service employs public-key cryptography. Specifically, a key pair can be used by each entity to sign information such as protocol requests and responses, and by anybody else to encrypt information meant to be communicated to the entity (e.g. in the case of peer negotiation as seen in Figure 2.1).

The inherent public key infrastructure problem is how different actors know each other's public keys. A typical implementation would be to establish a trusted authority that will sign certificates that map public keys to application-specific roles. However, this trusted authority becomes a central security and privacy point of failure. A central authority is still supported by PANORAMIX, but flexibility is needed so that applications themselves can establish the actor-to-key correspondence in a way that they know and guarantee is secure in their use-case.

As explained in D4.3 (Subsections 2.1.1), PANORAMIX peer authentication should offer peer-to-peer discovery of public keys, and through its negotiation mechanism, the mix network must establish consensus of the global peer list. To do so, the mechanism needs to use application-specific contacts. The basic information that actors or systems have before the establishment of a Public Key Infrastructure (PKI), provides the means to communicate privately and securely with each other (e.g. an email address, or a firewall-ed VPN connection). Hence, trust must be established by exchanging public keys through an initially available communication channel. The integrated service employs the PANORAMIX architecture and its APIs to facilitate such an exchange and key discovery.

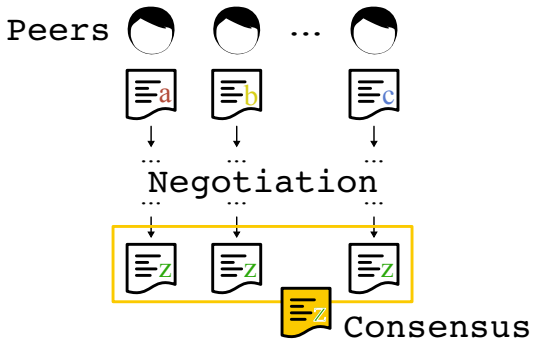

Figure 2.1: Public Cryptography is employed for Peer Negotiation in the e-voting Workflow of Zeus.

2.2 Usable and Secure Mix-contributor Configuration and Audit-Log for Administrators

A security-critical application like Zeus, distributes trust among network entities that are considered independent and are secured in different administrative domains. However, these entities need to communicate and collaborate in order to correctly execute an e-voting protocol and the coordination between them and its security implications are often overlooked.

A consensus over technical details during an election does not necessarily pre-exist (e.g. which cryptographic algorithms are used, what tuning parameters are used to initialize software, which are the actor's network endpoints and more). Very often, this consensus occurs because stakeholders are strongly interested in a specific mix-net. However, this interest makes those stakeholders well-known, prone to attacks, corruption and / or coercion. Our integrated service should offer an easy, yet trusted way for disinterested parties to be able to contribute.

There are several points to consider for this. First there is the *awareness of the consensus*. This means that human actors or automated entities must be able to review and explicitly agree on the consensus. *Usability* is another major requirement. We expect that users will not understand all parameters, but they have to be able to distinguish among different parameters and be able to relay information to expert consultants if they choose to do so. Finally, when considering that disinterested parties acquire responsibilities by being an actor in a cryptographic protocol such as a mix-net, *auditability* becomes a major requirement.

In the context of e-voting, the following observation is very prevalent: Since the disinterested actors have no knowledge of the process and application environment they will resist responsibilities in case they are manipulated by better informed but malicious stakeholders. To address this trend, our integrated service needs to offer: (1) ways to record the protocol requirements for each entity in the protocol so that their responsibilities are strictly defined, (2) an audit log of all configuration parameters and actions and (3) this audit log together with the system's safe defaults, can be used by the actors themselves to review their own security and to prove to investigating authorities that they did no wrong.

2.3 Integration Between a Mix-net Implementation and the PANORAMIX Controller

As we mentioned earlier, a software controller for authentication / PKI, configuration, and audit log management is an important contribution that PANORAMIX should bring to its users.

Nevertheless, as we have observed in Deliverable 5.2, this view requires that various mix-nets can be plugged in and be controlled through a modular interface. Hence, the PANORAMIX

controller software of the integrated service must provide an interface so that different mix-nets can be integrated. At the very least, integration means a module that encodes configuration parameters, actor roles, and can produce configuration files. At best, integration could mean that the mix-net is controllable at runtime.

3. PANORAMIX Integration with Zeus

3.1 Introduction

Zeus as a voting application cannot simply integrate a mix-net service as a simple component in the architecture without compromising on software quality and security. There are two reasons for that.

First, the mix-net dictates the cryptosystem that the rest of the application must use, i.e. the voting booth for the encryption of ballots, the ballot box for the validation of the submitted votes, the mixing chain for the validation of the mixes, and the trustee control panel for the distributed decryption of the mixed votes. Therefore, mixing has a central role in Zeus, not only regarding privacy, but also from a software engineering perspective.

Second, there is the issue of distributed trust. Placing trust in distributed independent actors is the only way to safeguard privacy and ensure proper observation of protocol, since central authorities can become coercible on their own or coercive themselves.

Unlike in other applications where the distribution of trust is set up at the time of the deployment of the service, in e-voting trust must be ideally distributed separately for each election event, as each event has their own stakeholders that have strong incentive to protect the proper procedure.

Other security and privacy-critical applications have complex initial set-up procedures (e.g. z-cash) or extensive legal and organizational overhead (e.g. financial clearing houses) to create an environment of trust. Zeus as an e-voting service has to create this environment over and over again as a matter of course for each individual election event, or category of events.

Therefore, the integration of Zeus with PANORAMIX involves the incorporation of modular cryptography using different mixing algorithms available, and the use of the negotiation and consensus mechanism to establish secure distributed control of the mixing process, and the e-voting process in general.

3.2 Architecture Overview

Figure 3.2 illustrates the architecture of the integrated system. The architecture is based on the PANORAMIX system described in Deliverable 4.3. Recall that the "Controller" of the system consists of three basic services, namely: registration, configuration and messaging. As depicted in the figure, each user runs the PANORAMIX client software and the software related to the corresponding mix-net.

Registration Service The registration service provides peer authentication and public key infrastructure which are critical in the context of e-voting. All entities in the mix-net must create their identity in the form of a public-private key pair and register it along with a mix-net-specific role designation. The service is then responsible to decide on and distribute a consistent view of the peers and roles in the network e.g. election trustees and more. The generation of identities and the assignment of roles are mix-net-specific actions, and may involve different configuration

steps as we explain later in the section.

Configuration Service The configuration service is responsible for the secure configuration and auditability by administrators and / or other parties. In an e-voting mix-net, different entities must independently agree on the initial setup or the current effective parameters. The agreement and configuration availability is critical for the function and security of the mix-net. An audit record of the agreements is also important for the auditability and accountability of the operation of the mix-net. The configuration service is designed to accept different proposals for an agreement under an arbitrary title. Then, actors negotiate through multiple rounds where they collect the proposals of others and submit their own. In the end, the service publishes the agreements in a secure manner.

Messaging Service After a mix-net has been set up and ready to work, this service accepts messages for delivery through the mix-net using a specific interface that is followed by the implementation. Furthermore, the service offers a message inbox where messages are queued for reading. Messages are typically sent and received by end-users. However, the messaging service may also be leveraged for internal communication among the mix-net entities.

3.3 Public Key Infrastructure and User Management

As mentioned in the introduction of this chapter, Zeus must use the same cryptography of the mix-net in use, therefore it must use PANORAMIX for managing public keys. Since Zeus also integrates with the negotiation and consensus facilities of PANORAMIX.

As described in Work Package 4 deliverables, a PANORAMIX peer is a basic entity identified by a public key and is able to send and receive messages, and negotiate and sign consensuses with other peers. In the integrated Zeus system, each important actor in the application workflow, such as an election trustee, or a mix-net contributor, and each important endpoint, such as the election ballot box, or a mixing server input, is represented by a PANORAMIX peer. Ultimately, the process is controlled by the trustees, who initiate the election event by jointly creating an election peer to serve as the control center of the whole process. The election peer signs a number of *standing orders* which are descriptions of what actions may be performed by actors in the application. There are also details on how long, how many times, and on what conditions such actions can take place. Typically, the election administrator is the recipient of most standing orders. Since the election peer is jointly owned by the trustees, every standing order requires a negotiation / consensus among them.

3.4 Workflow Integration

The general design of the integrated workflow is that all important actions must be permitted by a signed order. The actions are prepared and proposed, and before they are executed, they must either be permitted by an existing order, or they must be permitted by a newly issued order. Usually, the election administrator is the one who proposes actions while the election peer (i.e. the trustees) are the ones who permit it.

In the following, we list and describe the different actions that are prepared and signed for a Zeus election in its current form.

- CREATE ELECTION AUTHORITY: Prepared by an election administrator. It designates a committee of trustees as an election authority. It requires consensus of the trustees. It results in the creation of the election peer.
- CONFIGURE ELECTION EVENT: Prepared by an election administrator. Allows the creation of election events, including election type, polls, opening and closing times, ballot

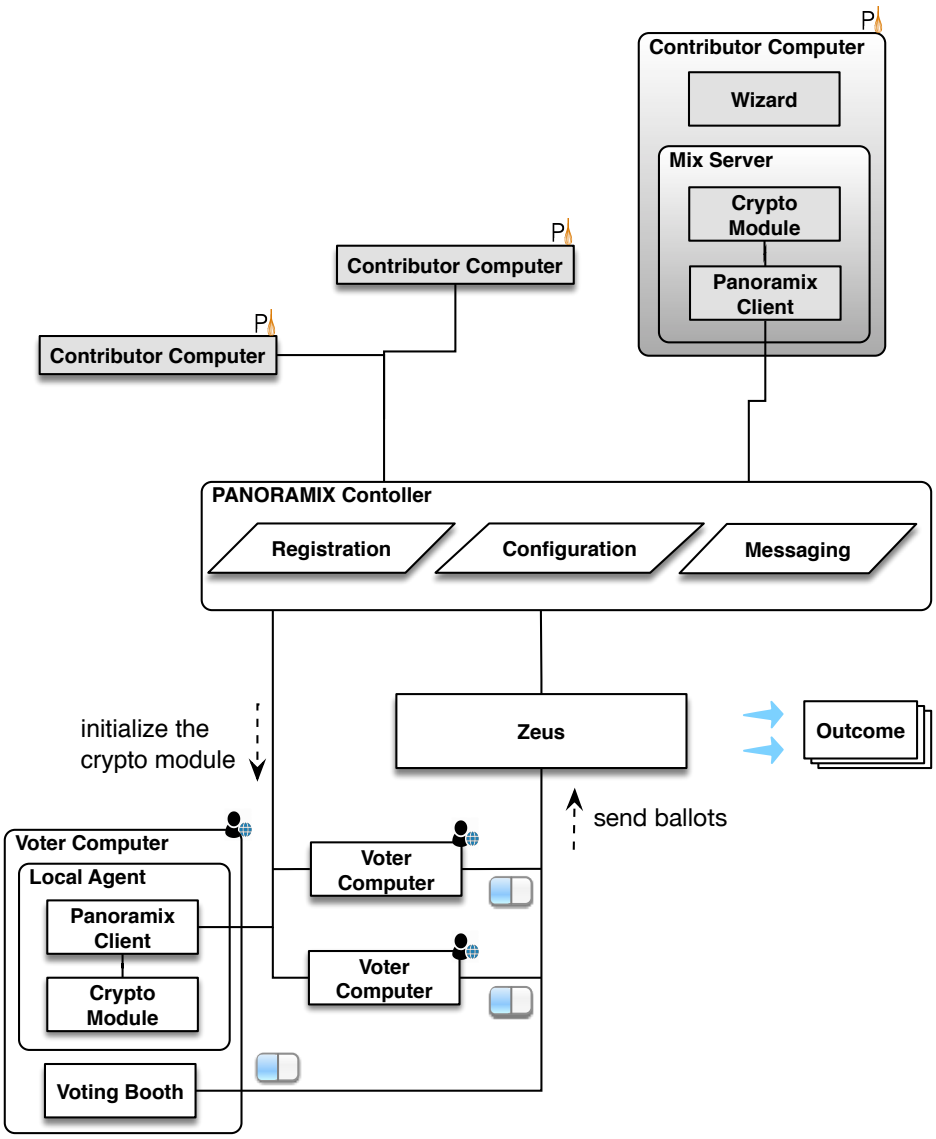

Figure 3.1: The Architecture of the Integrated System.

(candidate list), voter list, etc. The configured event cannot be finalized with this order (i.e. cannot freeze).

- FINALIZE ELECTION EVENT: Proposed by the administrator, this order allows the freezing of an election event. After this action is executed, the election event becomes officially active.
- MODIFY ELECTION EVENT: With this order, the election authority can modify an active election (e.g. voter list, times, extension). Every modification of a finalized (i.e. frozen) election must be approved by the election authority (i.e. committee of trustees). The order includes actions such as excluding voters, canceling, and closing the election. This order does not enable control of closed elections.
- NOTIFY VOTERS: The Administrator is allowed to send election event-related messages to the election voters and other stakeholders for a specified election. This ensures that communication is official and authoritative.
- MIX BALLOTS: After an election is closed, the administrator prepares the mixing by invit-

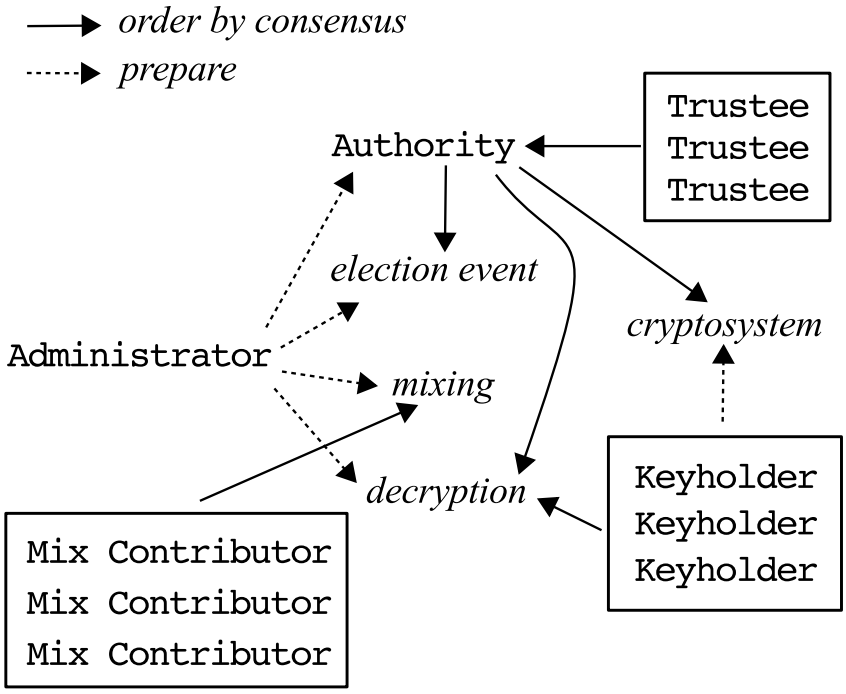

Figure 3.2: The General Workflow of our Integrated System.

ing mix contributors to mix the ballots one after another. Notably the mix contributors may be invited after election setup and there is no need to commit to their public key prior to election start. The mixed ballots can only be placed in the election dataset with a signed order from the mix contributors. There are two variants for this order, either each contributor signs their own mix, or, ideally, all contributors validate each other's mix round and sign a consensus order to place the entire mixing pipeline in the election.

Although the mix rounds are verifiable by anyone, it is a natural choice to have the mix contributors validate each other since they already have the cryptographic software, the computational resources, and the required technical expertise in order to provide a mix round in the first place.

- DECRYPT AND FINISH ELECTION: The administrator invites the key holders to take the final mixed ballots and provide partial decryptions. Again, anyone can verify the partial decryptions, however key holders are already capable and involved in the process and therefore it is natural to ask them to validate each other's decryptions and sign the final results in a consensus order.

4. Integrated mix-nets and Evaluation

As we have discussed in Deliverable 5.2, Zeus mixes votes with a Sako-Kilian re-encryption mix-net [5], based on the ElGamal cryptosystem [11]. A limiting factor in the wider adoption of Zeus and e-voting in general, is the mix-net implementation that is used by the voting platform. As mentioned above, the Zeus e-voting system currently uses a standard mix-net implementation that is simple, but not very efficient; only a few thousand voters can be handled within an acceptable amount of time (about an hour), while the needs of the system are in the order of tens or hundreds of thousands. Hence, we wanted to utilize the different mix-nets developed in PANORAMIX to make the integrated service more efficient.

We attempted to integrate two different mix-nets tailored for e-voting and we have evaluated our integrated service with positive results, The mix-nets integrated are: a *Bayer Groth* mix-net [2] and the re-encryption mix-net presented by Fauzi et al. [8]. Notably, the latter was designed and implemented in the context of PANORAMIX. In particular, this is a new non-interactive zero-knowledge proof (NIZK) [3] shuffle argument. In a standard zero-knowledge proof one party (the prover) can prove to another party (the verifier) that a given statement is true, without conveying any information apart from the fact that the statement is indeed true. NIZK proofs are a variant of zero-knowledge proofs in which no interaction is necessary between prover and verifier. This new shuffle argument has a simple structure, where the first verification equation ascertains that the prover has committed to a permutation matrix, the second verification equation ascertains that the same permutation was used to permute the ciphertexts, and the third verification equation ascertains that input ciphertexts were "correctly" formed.

We need to point out here that we have focused on the implementation of two mixes instead of putting all our effort into one because there are interesting differences between the two. For instance one avoids ROM usage and the other avoids pairings, while the latter still has a future speedup by means of switching to the `libsnark` library as we discuss in the following. In general, our initial tests indicate that the aforementioned mix-nets are faster than the Sako-Kilian mix-net that is currently employed by Zeus. We intend to further test them and use them in an election very soon. The implementation of the two mix-nets can be found at the following URLs:

```
https://github.com/grnet/hat_shuffle
https://github.com/grnet/bg-mixnet
```

4.1 Fauzi et al. re-encryption mix-net

The motivation behind this implementation is to replace the mix-net used by the e-voting application, Zeus in favor of a faster one. However, it can be used by anyone that needs a mix-net implementation. That is, apart from e-voting, the mix-net can be used for other tasks such as surveys and the collection of data from various IoT (Internet of Things) devices. The implementation was based on an existing prototype of the same re-encryption mix-net discussed in Deliverable 5.2.

The prototype implementation of the mix-net proposed by Fauzi et al., produces multiple

Metrics		
Operation	Short Description	Time per 100 voters
Initialization	Creates the elliptic Curve and private keys	364ms
Encryption	Encrypts the votes	674ms
Random Permutations	Creates random numbers	1ms
Proof	The shuffle	2085ms
Verification	Verification of the shuffle	2738ms
Decryption	Decrypts the votes	489ms

Table 4.1: Metrics

implementation difficulties. For implementations of cryptographic protocols it is typical to use C for informal cryptographic computations. Yet the prototype is implemented using Python, so it has to switch between Python and C for its operations. This may be a bottleneck of the prototype and the reason some operations are slower than they should be. Another reason may be that the underlying C cryptographic operations themselves are not efficient, and a different C implementation might improve matters. The two reasons are not exclusive, and one might compound the other. With our integration service at hand we managed to extensively evaluate the implementation in a way that we describe below.

Metrics Table 4.1 contains a list of metrics for the various operations of the prototype. Most of the time is taken by the prover and the verifier, as expected, because these have the most computations that produce a context switch between Python and C. The time taken by each of these operations is linear, meaning that for 200 ciphertexts the numbers on the table are doubled.

Context Switches A context switch happens when a Python program communicates with a C program for various cryptographic computations. The reasoning behind believing that a context switch may be the bottleneck of the application is that Python needs to create a PyObject containing the data it wants to communicate, and C also needs to create a PyObject to return the result of the computations.

The prover has various steps. In order to validate our theory about context switches we measured each of these steps. Two of those steps, while having the same number of iterations, had a significant time difference. In particular, `step2a` below took 100ms, while `step3a` took 700ms.

```
def step2a(sigma, A1, randoms, g1_poly_zero, g1rho, g1_poly_squares):
    pi_1sp = []
    inverted_sigma = inverse_perm(sigma)
    for inv_i, ri, Ai1 in zip(inverted_sigma, randoms, A1):
       g1i_poly_sq = g1_poly_squares[inv_i]
        v = (2 * ri) * (Ai1 + g1_poly_zero) - (ri * ri) * g1rho + g1i_poly_sq
        pi_1sp.append(v)
    return pi_1sp

def step3a(sigma, ciphertexts, s_randoms, pk1, pk2):
   v1s_prime = []
    v2s_prime = []
   for perm_i, s_random in zip(sigma, s_randoms):
        (v1, v2) = ciphertexts[perm_i]
        v1s_prime.append(tuple_add(v1, enc(pk1, s_random[0], s_random[1], 0)))
```

```
        v2s_prime.append(tuple_add(v2, enc(pk2, s_random[0], s_random[1], 0)))
    return list(zip(v1s_prime, v2s_prime))
```

First we attributed the time difference to various calls to zip and to tuple creation. After removing all the calls to zip we didn't notice any significant difference. This seemed to validate the context switches theory, because the slower step contained more context switches per iteration.

However that is not the case. Using cProfile we identified the main reason behind this difference. The slower step does more multiplications on elliptic curve elements. While it is expected that multiplication will be slower than addition, the difference was enough to dismiss the context switches theory.

Multiplication on our elliptic curve elements takes 575ms per 300 multiplications, while addition takes 5ms for 400 additions. If the real problem was context switches, then addition wouldn't have such a huge difference with multiplication, because it has more operations hence more context switches.

Comparing bplib and libsnark The prototype implementation uses the `bplib`[7] Python module. `bplib` implements bilinear pairings on elliptic curves while also supporting elliptic curve operations using the openssl library. Another implementation supporting elliptic curve computations and bilinear pairings is `libsnark`[9]. The common characteristics of these libraries are that they both use the Ate Pairing and they use windowed exponentiation for optimization purposes.

A key difference of these implementations is that `bplib` uses the *curve Fp254BNb*, while `libsnark` uses *bn128* which is a patch on the *Fp254BNb curve*. Also, `libsnark` supports vectorized exponentiation which boosts its performance.

In order to compare these two libraries and test our theory that `libsnark` is faster than `bplib`, we created two different tests using `bplib` and `libsnark` on each one. The tests did multiplications (the bottleneck of the prototype) on both elliptic curve groups.

The results validated our theory. Multiplying elements on the $\mathbb{G}_2$ group using libsnark yielded a performance of 0.38s/1000 ciphertexts while using openSSL yielded 3.22s/1000. On G1 libsnark produced 0.13s/1000 while `bplib` produced 0.96s/1000 multiplications.

Wrapping libsnark with Cython Since `libsnark` computes multiplications faster than the openssl implementation, the most obvious solution is to replace openssl with `libsnark`. The best candidate for this job is Cython, because it offers the performance aspects of C, while providing the functionality of Python. In order to validate that Cython, indeed will yield better performance we created a basic Cython application that does multiplications on the $\mathbb{G}_2$ group of the `libsnark` elliptic curve.

The results were positive. A multiplication on the $\mathbb{G}_2$ group using Cython takes about 0.5ms while on our prototype implementation, that uses `bplib`, a multiplication takes about 2ms. So that's a 4x boost in performance. Also our Cython implementation didn't implement vectorized multiplications, so there's still room for optimizations.

Solutions Since the real bottleneck are the multiplications on G2 elements, the most obvious solution is to use optimizations on the multiplication process.

As mentioned, `libsnark` computes multiplications faster than our current implementation. Yet `libsnark` is written in C++ and we want to use a Python module. A Python wrapper for `libsnark` would be useful, for our needs and the open source community. In fact, we do not really need the full `libsnark` library; the Elliptic curve parts have been factored out to libff [10], so a Python wrapper for libff could be implemented.

4.2 Bayer Groth mix-net

The Bayer Groth mix-net has been a focus of many research papers (e.g. the Stadium project [13]). In our implementation we are focusing on fine-tuning instead of starting the implementation from scratch. In our integration, we particularly modified the mix-net's verifiable shuffle, decreasing latency by more than an order of magnitude. Specifically, we optimized the shuffle by applying the following improvements:

- We have added OpenMP directives[1] to optimize key operations, such as Brickell et al.'s multi-exponentiation routines [4].
- We replaced the use of integers with Moon and Langley's implementation of Bernstein's *curve25519* [6] group (note that we avoid point compression and decompression in intermediary operations to improve speed).
- We improved point serialization and deserialization with byte-level representations of the data.
- Taking into account different performance profile of *curve25519*, we replaced some multi-exponentiation routines with naive version and tweaked multi-exponentiation window sizes. The bottleneck for the shuffle is currently in multi-exponentiation routines.
- Finally, we added some more small optimizations (e.g. powers of 2, reduce dynamic memory allocations, etc.).

[1] https://msdn.microsoft.com/en-us/library/0ca2w8dk.aspx

5. Conclusions

In this deliverable, we discussed the integrated service developed in the context of Work Package 5. Through this service we have fulfilled all the remaining requirements regarding e-voting, enumerated in Deliverable 5.2. To validate the service we have integrated two new mix-nets. Notably, one of the two was designed and implemented in the context of the project, as joint work by the University of Tartu and GRNET. Our service also lets us perform extensive evaluation experiments on the mix-nets.

In general, our initial aim was to deliver an e-voting service supporting large scale elections in an effective way. As we discussed in this deliverable, we have designed the integrated service to be built on top of the mix-net infrastructure developed in Work Package 4. The whole process is verifiable end-to-end; from the encryption of ballots on the voter's device, through the mix-net service, and back to the e-voting service for counting. Voters are able to verify that their vote was counted in the results and election authorities will have access to proof for the correctness of the process.

Bibliography

[1] The European Union general data protection regulation. http://data.consilium.europa.eu/doc/document/ST-5419-2016-INIT/en/pdf, 2016. [Online; accessed 30-November-2017].

[2] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'92, pages 263–280, Berlin, Heidelberg, 2012. Springer-Verlag.

[3] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 103–112, New York, NY, USA, 1988. ACM.

[4] Ernest F. Brickell, Daniel M. Gordon, Kevin S. McCurley, and David B. Wilson. Fast exponentiation with precomputation. In *Proceedings of the 11th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'92, pages 200–207, Berlin, Heidelberg, 1993. Springer-Verlag.

[5] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, February 1981.

[6] Yu-Fang Chen, Chang-Hong Hsu, Hsin-Hung Lin, Peter Schwabe, Ming-Hsien Tsai, Bow-Yaw Wang, Bo-Yin Yang, and Shang-Yi Yang. Verifying curve25519 software. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 299–309, New York, NY, USA, 2014. ACM.

[7] George Danezis. A bilinear pairing library for petlib. https://github.com/gdanezis/bplib.

[8] Prastudy Fauzi, Helger Lipmaa, and Michal Zajac. An efficient pairing-based shuffle argument. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Application of Cryptology and Information Security, 2017*, 2017.

[9] SCIPR Lab. A C++ library for zkSNARK proofs. https://github.com/scipr-lab/libsnark.

[10] SCIPR Lab. libff: C++ library for finite fields and elliptic curves. https://github.com/scipr-lab/libff.

[11] Bruce Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[12] Georgios Tsoukalas, Kostas Papadimitriou, Panos Louridas, and Panayiotis Tsanakas. From Helios to Zeus. In *2013 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '13, Washington, D.C., USA, August 12-13*, 2013.

[13] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nickolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 423–440, New York, NY, USA, 2017. ACM.