Anonymisation: Problems and Solutions
From Theory to Practice

Associate Professor Panos Louridas

Athens University of Economics and Business

Overview

Overview

1. Basic Stuff
2. Decryption Mixnets
3. Requirements for an e-Voting System
4. The Zeus Voting System and Process
5. Re-encryption Mixnets

Diffie-Hellman Key Exchange

<table>
<tr><th>Alice</th><th>Bob</th></tr>
<tr><td colspan="2">Alice and Bob agree on p and g</td></tr>
<tr><td>Choose a
Calculate $A = g^a \bmod p$
Send A to Bob</td><td>Choose b
Calculate $B = g^b \bmod p$
Send B to Alice</td></tr>
<tr><td>Calculate
$s = B^a \bmod p$
$\quad = (g^b)^a \bmod p$
$\quad = g^{ba} \bmod p$</td><td>Calculate
$s = A^b \bmod p$
$\quad = (g^a)^b \bmod p$
$\quad = g^{ab} \bmod p$</td></tr>
</table>

What are a, b, g and p?

- g and p are such that g is a generator of a finite cyclic group G of order p, e.g., $G = (\mathbb{Z}_g)^*$ of order p.
- In plain words:

$$G = \{1, g, g^2, \ldots, g^{p-1}\}$$

- Then we have:

$$a \in G$$

$$b \in G$$

Diffie-Hellman Security

- There is no known efficient way to find the secret from p, g, A, and B.
- That is because to do that we would need to solve the *discrete logarithm problem*, for which we have no efficient solution.
- If p is prime, and we have g and $y = g^x \bmod p$, the discrete logarithm problem is finding x, $1 \leq x \leq p - 1$.
- The integer x is called *discrete logarithm of y with base g* and we write $x = \log_g y \bmod p$.

The ElGamal System

All calculations are modulo p.

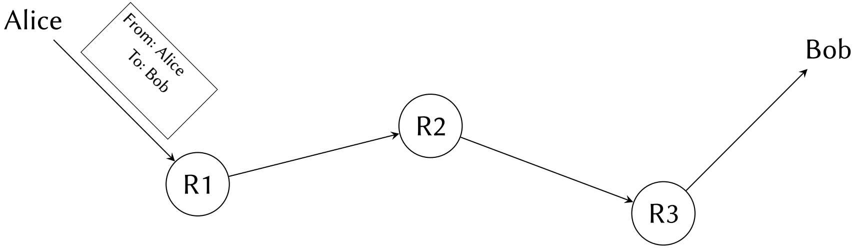
Alice	Bob
Choose a Calculate $A = g^a$ A, p, g form Alice's public key.	Choose b Calculate $c_1 = g^b$ Calculate $s = A^b = g^{ab}$ Calculate $c_2 = m \cdot s = mg^{ab}$ Send $(c_1, c_2) = (g^b, mg^{ab})$ to Alice
Calculate $s = c_1^a = g^{ab}$ Decrypt with $c_2 s^{-1} =$ $mg^{ab}(g^{ab})^{-1} =$ $m(g^{ab}g^{-ab}) = m$	

ElGamal Re-encryption

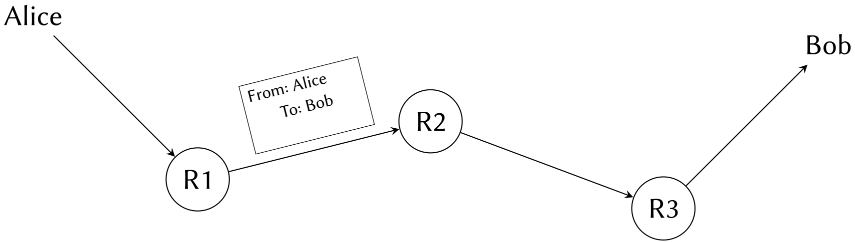
- Say that Bob creates a ciphertext $c = (c_1, c_2)$ and sends it to Charlie.
- Charlie selects $s' \in G$.
- Charlie computes $c' = (c'_1, c'_2) = (g^{s'} c_1, g^{as'} c_2)$.
- That is, Charlie *re-encrypts* the ciphertext.
- Alice then can calculate $c_1'^a = (g^{s'} c_1)^a$ and then $((g^{s'} c_1)^a)^{-1} g^{as'} c_2 = g^{-s'a} c_1^{-a} g^{as'} c_2 = c_1^{-a} c_2 = g^{-ab} m g^{ab} = m$
- Which means that if Bob encrypts a text, and then Charlie re-encrypts it, Alice can still decrypt it, even without knowing Charlie's key!

Overview

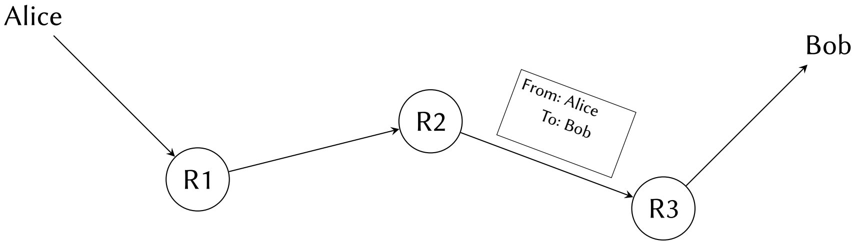
Message Traveling from Alice to Bob (1)



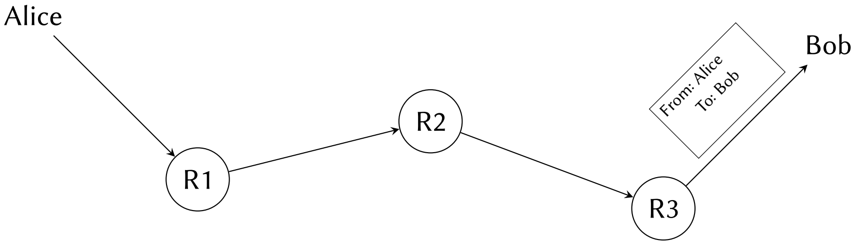
Message Traveling from Alice to Bob (2)



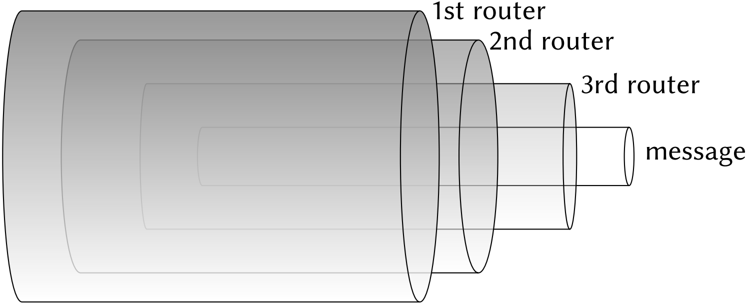
Message Traveling from Alice to Bob (3)



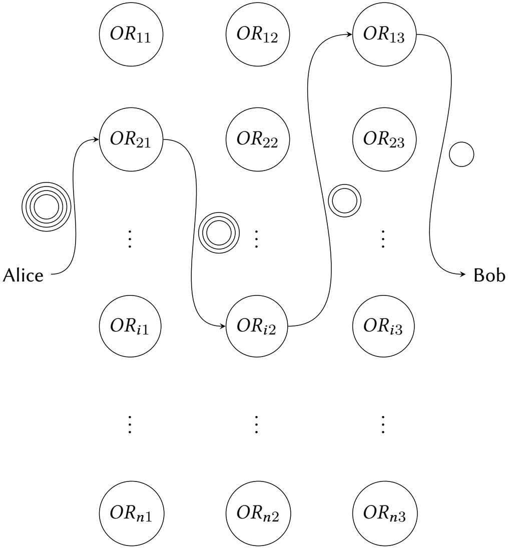
Message Traveling from Alice to Bob (4)



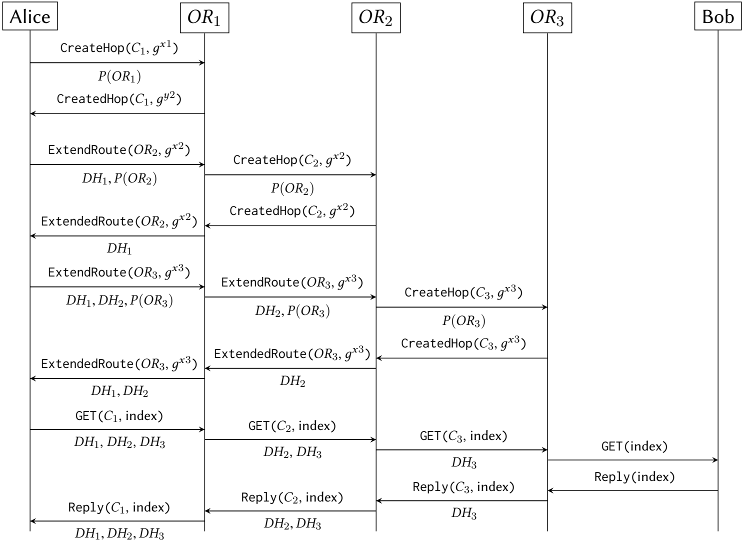
Onion Routing



TOR: The Onion Router



Onion Routing in Tor



Tor Setup (1)

- Alice communicates with OR_1 using RSA and sending instructions on how to set up the communications routing.
- This is a *command packet* containing her part of a Diffie-Hellman key exchange with OR_1.
- In addition, it contains a command that tells OR_1 that she will be tagging her packets with a special ID she picks, called a *circuit id*, say C_1.
- Let us call this command packet a `CreateHop`(C_1, g^{x1}) abbreviating the Diffie-Hellman part in our notation.
- OR_1 replies with its part of the Diffie-Hellman key exchange.
- All messages that Alice will be sending to OR_1 will be encrypted with the key they established, say DH_1.

Tor Setup (2)

- Next Alice communicates again with OR_1 and tells it that from now on she wants OR_1 to forward all messages from her to OR_2.
- To do that she sends a command packet to OR_1 with the command to extend the route and her part of a new Diffie-Hellman key exchange.
- The Diffie-Hellman part is encrypted with the RSA public key of OR_2.
- The whole packet is encrypted with DH_1. Let us call that command packet an `ExtendRoute`(OR_2, g^{x2}) packet.

Tor Setup (3)

- When OR_1 gets the packet it decrypts it. It then creates a new `CreateHop(`C_2, g^{x2}`)` packet that it sends to OR_2.
- The command packet contains the Diffie-Hellman part from Alice to OR_2, and tells OR_2 that it will be tagging packets with another circuit ID, say C_2.
- It tells that to OR_2, without telling it that the messages will be coming from Alice.
- OR_1 records the fact that packets tagged with C_1 will be sent to OR_2, and packets received from OR_2 tagged with C_2 will be passed back to Alice.
- OR_1 passes back the Diffie-Hellman response it receives from OR_2 to Alice, so Alice and OR_2 share a Diffie-Hellman key, DH_2.

Tor Setup (4)

- To create the route to OR_3, Alice creates an `ExtendRoute`(OR_3, g^{x3}) command packet to extend the route from OR_2 to OR_3.
- The packet contains her part of a Diffie-Hellman key she wants to establish with OR_3.
- The Diffie-Hellman part is encrypted with the RSA public key of OR_3.
- The whole packet is encrypted with DH_2 and then encrypted on top with DH_1.
- Alice sends the packet to OR_1. When OR_1 gets the packet, it is able to decrypt the first layer only.
- OR_1 knows that cells tagged with C_1 must be forwarded to the destination associated with C_2, OR_2, but it does not know its contents. It tags the packet with C_2 and forwards the packet with one layer peeled off to OR_2.

Tor Setup (5)

- OR_2 gets the packet from OR_1 and decrypts it using DH_2, retrieving `ExtendRoute(`OR_3, g^{x3}`)`.
- It creates and sends a new command packet `CreateHop(`C_3, g^{x3}`)` to OR_3. The command packet contains the Diffie-Hellman part from Alice to OR_3 and tells it that it will be tagging packets with another circuit ID, say C_3.
- OR_2 records the fact that packets tagged with C_2 will be sent to OR_3, and packets received from OR_3 tagged with C_3 will be passed back to OR_1.
- OR_2 passes back the Diffie-Hellman response from OR_3 to Alice via OR_1, so Alice and OR_3 share a Diffie-Hellman key, DH_3.

Tor Messaging from Alice to Bob (1)

- To send a message to Bob, Alice creates a packet with her message addressed to Bob encrypted with DH_3, in turn encrypted with DH_2, in turn encrypted with DH_1 and tagged with C_1.
- The packet goes first to OR_1. Because the packet is tagged with C_1, OR_1 knows it must forward it to OR_2.
- OR_1 peels off the first layer using DH_1 and forwards it to OR_2, tagged with C_2.
- OR_2 peels off the second layer using DH_2. It knows that packets tagged with C_2 must be forwarded to OR_3, so it tags it with C_3 and sends it to OR_3.

Tor Messaging from Alice to Bob (2)

- OR_3 gets the packet from OR_2 and decrypts it using DH_3.
- It sees that it is a message addressed to Bob, so it just forwards it there.
- The response from Bob will follow exactly the reverse route, Bob $\rightarrow OR_3 \rightarrow OR_2 \rightarrow OR_1 \rightarrow$ Alice,
- It will be encrypted again with DH_1, then DH_2, then DH_3, routed in the same way using C_3, C_2, C_1.

Overview

Democratic

- Only eligible voters can vote.
- Each eligible voter can cast at most one vote (that counts).

Source of e-Voting System Requirements: `http://courses.csail.mit.edu/6.897/spring04/L17.pdf`

Private

- No one can tell how a voter actually voted (anonymity, at least within large enough cohort/precinct of voters).
- OK (perhaps even mandatory) to publish who voted (though, obviously not actual ballot content).

Uncoercible

- Voter cannot be coerced or bribed to vote a particular way.
- Voter cannot prove how they voted to another party: receipt-free. (Note how this requirement assumes the voter may be an adversary.)

Accurate

The final tally is the correct sum of cast votes.

- Cast ballots can't be altered, deleted, substituted.
- All cast ballots are counted; other (invalid) ballots can't be added.

Verifiable

> *I consider it completely unimportant who in the party will vote, or how; but what is extraordinarily important is this—who will count the votes, and how.*

Joseph Stalin
In Russian: Я считаю, что совершенно неважно, кто и как будет в партии голосовать; но вот что чрезвычайно важно, это—кто и как будет считать голоса.
Said in 1923, as quoted in The Memoirs of Stalin's Former Secretary (1992) by Boris Bazhanov [Saint Petersburg] (Борис Бажанов. Воспоминания бывшего секретаря Сталина).
Variant (loose) translation: The people who cast the votes decide nothing. The people who count the votes decide everything.

Verifiable

- Individual verifiability: each voter may verify their vote.
- Representative verifiability: each voter may delegate to a party or other representative the task of verifying the vote (without revealing the vote in the clear, of course).
- Universal verifiability: anyone can verify total.

Robust

- A small group can't disrupt election (DOS attacks, complaint procedures, ...).

Fairness

- No partial results are known before the election is closed.

Ease of Use

- Good user interface.
- Accessibility and usability guidelines.
- Accessibility from a wide variety of input devices.

Efficiency

- Efficient ballot casting.
- Efficient ballot counting.

Overview

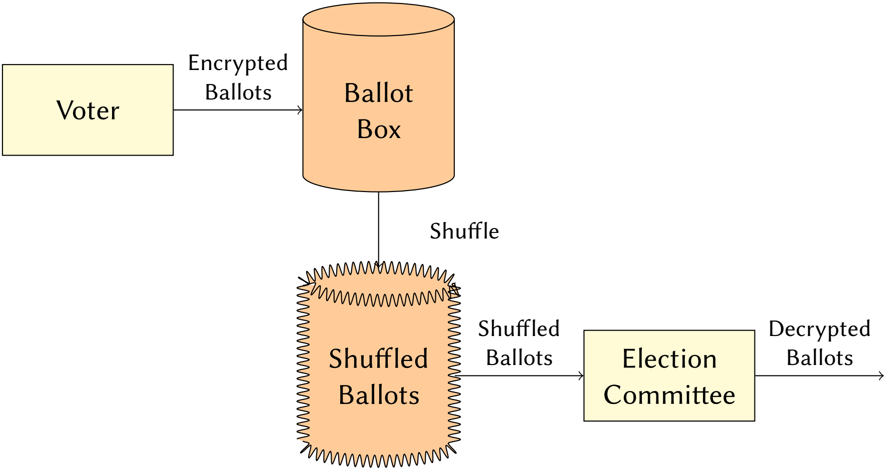
Zeus

- Zeus is an online voting system that lives on `https://zeus.grnet.gr`.
- It is based on Helios, a verifiable online elections system since 2008.
- Open source `http://heliosvoting.org/`.
- In particular, version 1 of Helsion used as the basis of Zeus.

Zeus is described in G. Tsoukalas, K. Papadimitriou, P. Louridas, P. Tsanakas, *From Helios to ZEus*, USENIX Journal of Election Technology and Systems (JETS), 1(1), 2013.

For Helios, see Ben Adida, *Helios: web-based open-audit voting*. In Proceedings of the 17th conference on Security symposium (SS'08), USENIX Association, Berkeley, CA, USA, pages 335–348.

Election Workflow



Basic Ideas

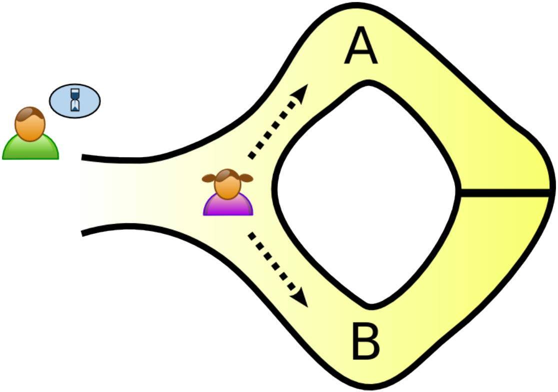
- Ballots are encrypted on the browser before being sent to the server.
- Ballots are stored in the server in encrypted form.
- The decryption keys are kept by the Election Committee.
- Encrypted ballots are randomly mixed in order to break the association between ballots and voters.
- The encrypted mixed ballots are decrypted by the Election Commitee.
- The process can be verified mathematically.

Basic Assumptions

- You do not need to trust the administrators of Zeus.
- You do not need to trust each member of the Election Committee.
- You need to trust that at least one member of the Election Committee is honest.
- Coercion is avoided by allowing multiple ballots per user (only the last one counts).

Overview

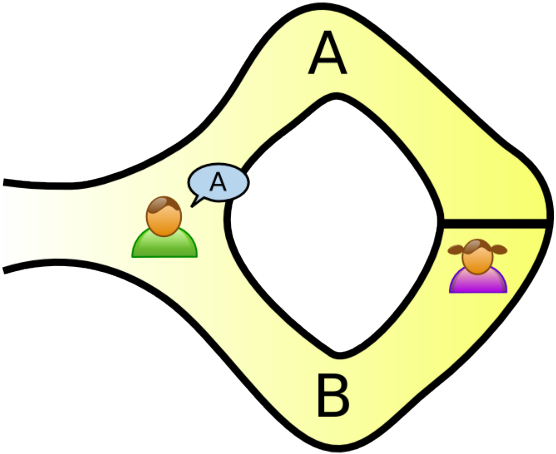
Zero Knowledge Proofs



Zero-Knowledge Proof.

Source: `http://en.wikipedia.org/wiki/Zero-knowledge_proof`

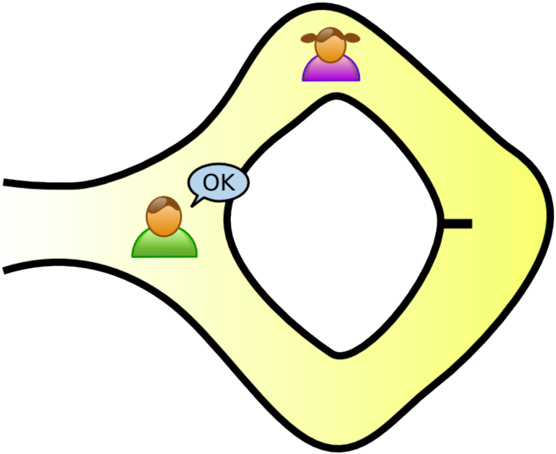
Zero Knowledge Proofs



Zero-Knowledge Proof.

Source: `http://en.wikipedia.org/wiki/Zero-knowledge_proof`

Zero Knowledge Proofs



Zero-Knowledge Proof.

Source: `http://en.wikipedia.org/wiki/Zero-knowledge_proof`

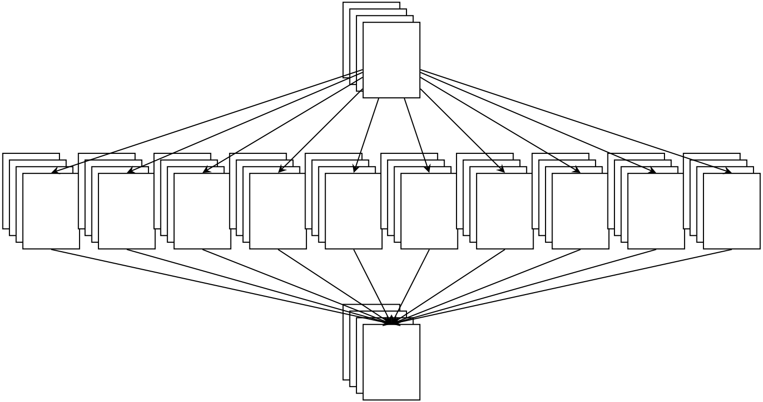
Re-encryption Mixnet and Zero Knowledge

- We must be able to prove to a verifier that the shuffle is honest.
- The verifier must not be able to gain any knowledge from that, apart from the fact that the shuffle is correct.

Basic Idea

- We take the encrypted messages and we re-encrypt them.
- We shuffle the re-encrypted messages and return the shuffled re-encrypted messages.
- Somehow we have to prove that the messages are the same, without revealing the permutation.

Re-encryption Mixnet Diagram



- A shuffle is really a permutation.
- Using Cauchy's notation for permutations, a shuffle of five items would be:

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 4 & 3 & 1 \end{pmatrix}$$

How to Prove the Shuffle

- The prover has the original ciphertexts.
- The prover creates the shuffle, along with a number of *shadow shuffles*.
- Suppose the prover has n shadow shuffles.

How to Verify the Shuffle (1)

The verifier asks the prover one of the following questions, for each $1 \leq i \leq n$:

- "For shuffle i, show me how to get from the original set of ciphertexts to i".
- "For shuffle i, show me how to get from i to the final shuffle".

How to Verify the Shuffle (2)

- Say the prover has performed the shuffle:

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 4 & 3 & 1 \end{pmatrix}$$

- And also the shadow shuffle:

$$\sigma' = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 5 & 2 & 4 \end{pmatrix}$$

- Then the prover can derive the following shuffle, which shuffles σ' to σ.

$$\sigma'' = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 3 & 2 & 1 & 4 \end{pmatrix}$$

How to Verify the Shuffle (3)

- The verifier can ask the prover to reveal the s' shuffle or the s'' shuffle *but not both.*
- If the prover is honest, they can always reveal what is asked from them.
- A cheating prover can provide at most one of s' or s''.
- So the cheating prover has 50% chances of cheating undetected.

Sake-Kilian Shuffle

- The above is the idea behind the Sako-Kilian mixnet.
- The prover performs n shadow shuffles.
- The probability that the prover can cheat without being detected is 2^{-n}.
- We choose n high enough to make it improbable.

For more details on the Sako-Kilian mixnet, see Kazue Sako and Joe Kilian, *Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth*, EUROCRYPT '95, Springer-Verlag, Berlin, Heidelberg, pages 393–403.

Non-Interactive Proof

- However, we want these challenges and answers to be non-interactive.
- To do this, we apply the Fiat-Shamir heuristic.
- The questions are determined by *challenge bits*.
- The challenge bits are derived from hashes of the shadow mixes.

For the Fiat-Shamir heuristic, see Amos Fiat and Adi Shamir, *How to prove yourself: Practical solutions to identification and signature problems*, CRYPTO '86, volume 263 of Lecture Notes in Computer Science, pages 186–194, Springer, 1986.

Fiat-Shamir in Practice

- We take a SHA-256 hash of the mixed ciphertexts for all mixes.
- We then read the bits of the result hex digest of the hash, and use them as challenge.
- In this way the prover is committed to the shuffles.

The Problem with Sako-Kilian

- The problem with the Sako-Kilian mixnet is that it requires a lot of computation.
- It needs a lot of shuffles, and each shuffle needs a lot of cryptographic operations.
- Therefore, to be able to handle millions of votes efficiently we need another mixnet system.
- There has been a lot of research of mixnets.
- Unfortunately, there have also been a lot of patents in mixnets.

A New Shuffle Argument

- A new, efficient patent-free mixnet is described in Prastudy Fauzi, Helger Lipmaa and Michał Zając, *A Shuffle Argument Secure in the Generic Model*, ASIACRYPT (2) 2016, volume 10032 of Lecture Notes in Computer Science, pages 841–872, Springer, Heidelberg. `https://eprint.iacr.org/2016/866.pdf`.
- The mixnet is based on elliptic curve cryptography and bilinear mappings, which we'll see in a bit.

A New Shuffle Argument

$\mathsf{gencrs}(1^\kappa, n \in \mathrm{poly}(\kappa))$**:** 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, chosen in Sect. 5. Set $\boldsymbol{\chi} = (\chi, \alpha, \varrho, \beta, \gamma) \leftarrow_r \mathbb{Z}_q^2 \times (\mathbb{Z}_q \setminus \{0\})^2 \times (\mathbb{Z}_q \setminus \{0, -1\})$. Let enc be the lLin cryptosystem with the secret key γ, and let $(\mathsf{pk}_1, \mathsf{pk}_2)$ be its public key. Set

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

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

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

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 $(\mathfrak{v}'_{i1}, \mathfrak{v}'_{i2}) \leftarrow (\mathfrak{v}_{\sigma(i)1}, \mathfrak{v}_{\sigma(i)2}) \cdot (\mathsf{enc}_{\mathsf{pk}_1}(0; \boldsymbol{s}_i), \mathsf{enc}_{\mathsf{pk}_2}(0; \boldsymbol{s}_i))$.
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) $(\boldsymbol{\pi}_{\mathsf{c2}:1}, \boldsymbol{\pi}_{\mathsf{c2}:2}) \leftarrow \prod_{i=1}^{n}(\mathfrak{v}_{i1}, \mathfrak{v}_{i2})^{r_i} \cdot (\mathsf{enc}_{\mathsf{pk}_1}(0; \boldsymbol{r}_s), \mathsf{enc}_{\mathsf{pk}_2}(0; \boldsymbol{r}_s))$.
7. Return $\pi_{sh} \leftarrow (\mathfrak{v}', (\mathfrak{A}_{i1}, \mathfrak{A}_{i2})_{i=1}^{n-1}, (\pi_{\mathsf{1sp}:i})_{i=1}^n, \pi_{\mathsf{c1}:1}, \pi_{\mathsf{c1}:2}, \boldsymbol{\pi}_{\mathsf{c2}:1}, \boldsymbol{\pi}_{\mathsf{c2}:2})$.

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

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 $\mathfrak{R} \leftarrow \hat{e}(\hat{\mathfrak{g}}_1, \pi_{\mathsf{c1}:2}^{p_{42}}(\pi_{\mathsf{c1}:1}\pi_{\mathsf{c1}:2})^{p_{43}}) \cdot \hat{e}(\mathfrak{h}_1, \pi_{\mathsf{c1}:1}^{p_{41}}\pi_{\mathsf{c1}:2}^{p_{42}}) / \hat{e}\left(\prod_{j=1}^3 \pi_{\mathsf{c2}:1j}^{p_{4j}}, \mathfrak{g}_2^{\varrho}\right)$.
6. Check that `/* Consistency: */`
   $$\prod_{i=1}^n \hat{e}\left(\prod_{j=1}^3 (\mathfrak{v}'_{i1j})^{p_{4j}}, \mathfrak{g}_2^{P_i(\chi)}\right) / \prod_{i=1}^n \hat{e}\left(\prod_{j=1}^3 \mathfrak{v}_{i1j}^{p_{4j}}, \mathfrak{A}_{i2}\right) = \mathfrak{R}.$$

From Theory to Reality

- Our task is to make a working system out of the above description.
- It turned out that moving from theory to reality was far more interesting that we would have anticipated.

Elliptic Curves

Definition

The elliptic curve over $\mathbb{Z}_p$, $p > 3$, is the set of all pairs $(x, y) \in \mathbb{Z}_p$ so that:

$$y^2 = x^3 + ax + b \bmod p$$

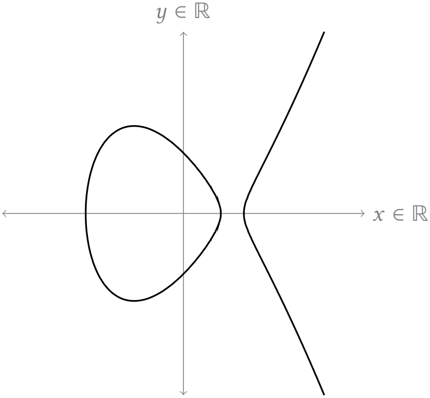
together with an imaginary point at infinity $\mathcal{O}$, where

$$a, b \in \mathbb{Z}_b$$

and

$$4a^3 + 27b^2 \neq 0 \bmod p$$

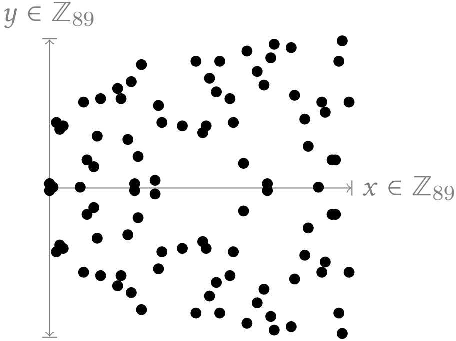
$y^2 = x^3 - 2x + 1$ over $\mathbb{R}$



$$y^2 = x^3 - 2x + 1 \text{ over } \mathbb{R}$$

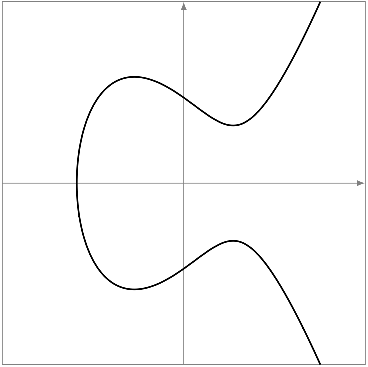
This and the following plots are adapted from Jérémy Jean, TikZ for Cryptographers, `http://www.iacr.org/authors/tikz/`, 2016.

$y^2 = x^3 - 2x + 1$ over $\mathbb{Z}_{89}$



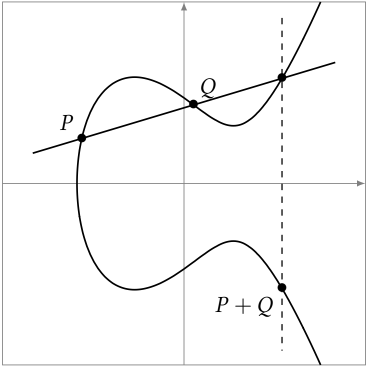
$$y = x^3 - 2x + 1 \text{ over } \mathbb{Z}_{89}$$

Elliptic Curve $y^2 = x^3 + 2x - 2$



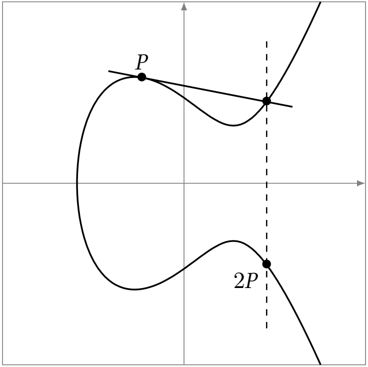
$$y^2 = x^3 + 2x - 2$$

Elliptic Curve Addition



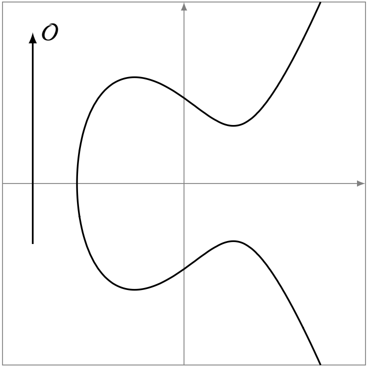
Addition $P + Q$
"Chord rule"

Elliptic Curve Doubling



Doubling $P + P = 2P$
“Tangent rule”

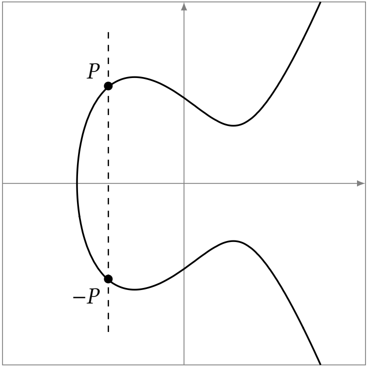
Elliptic Curve Point at Infinity



Neutral element $\mathcal{O}$

$$P + \mathcal{O} = P, P - P = \mathcal{O}$$

Elliptic Curve Inverse



Inverse element $-P$

Elliptic Curve Addition and Doubling

$$x_3 = s^2 - x_1 - x_2 \bmod p$$

$$y_3 = s(x_1 - x_3) - y_1 \bmod p$$

where

$$s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \bmod p, & \text{if } P \neq Q \text{ (point addition)} \\ \frac{3x_1^2 + a}{2y_1} \bmod p, & \text{if } P = Q \text{ (point doubling)} \end{cases}$$

s is the slope of the line passing through P and Q (in addition) or the slope of the tangent through P (in doubling).

From Integers to Elliptic Curves and Back

- If n is an integer, to find the corresponding point, we add the generator (point 1) n times.
- The reverse operator is not that straightforward.
- Unfortunately, there is not direct mapping *back* from an elliptic curve to an integer.
- That means that we have to use lookup tables or some other search-based methods.

Additive Notation

- Let G be a group of prime order r.
- The notation $[a]P$ corresponds to scalar multiplication of a generator $P \in G$ by a scalar $a \in \mathbb{Z}_r$.
- That is:
  $[a]P = P + P + \cdots + P$, a times when $a > 0$, or
  $[a]P = -P - P - \cdots - P$, a times when $a < 0$.
- We will use 0_G as the neutral element of group G.
- If G is a multiplicative group of prime order r, we will use 1_G as the neutral element of G.

ElGamal Encryption Revisited

We have a message $m \in G$ and a public key $(P, Y) \in G^2$. G is a group of prime order r and P is a generator of G.

1. The secret key is $x \in \mathbb{Z}_r^*$ and the public key is $Y = [x]P$.
2. Choose $\rho \in \mathbb{Z}_r^*$ at random.
3. Compute $T_1 = m + [\rho]Y$ and $T_2 = [\rho]P$.
4. Output $C = (T_1, T_2)$.

ElGamal Decryption Revisited

Output:

$$T_1 - [x]T_2 = m + [\rho]Y - [x][\rho]T_2 = m + [\rho][x]P - [x][\rho]P = m$$

Multiplicative Notation

- Let G be a group of prime order r.
- The notation P^a corresponds to exponentiation of a generator $P \in G$ by a scalar $a \in \mathbb{Z}_r$.
- That is:
  $P^a = P \times P \times \cdots \times P$, a times when $a > 0$, or
  $P^a = P^{-1} \times P^{-1} \times \cdots \times P^{-1}$, a times when $a < 0$.
- We will use 1_G as the neutral element of group G.

Bilinear Pairings

Definition

A *bilinear pairing* on (G_1, G_2, G_T), where G_1 and G_2 are groups with additive notation and G_T is a group with multiplicative notation, all of prime order r, is a map

$$\hat{e} : G_1 \times G_2 \longrightarrow G_T$$

with the following properties:

1. *Bilinearity*: For all $P_1 \in G_1$, $P_2 \in G_2$, and $a, b \in \mathbb{Z}_r$, we have:

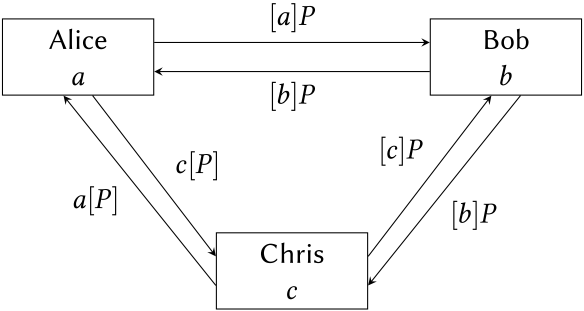
$$\hat{e}([a]P_1, [b]P_2) = \hat{e}(P_1, P_2)^{ab}$$

2. *Non-degeneracy*: for $P_1 \neq 0_{G_1}$ and $P_2 \neq 0_{G_2}$, $\hat{e}(P_1, P_2) \neq 1_{G_T}$.
3. *Computability*: $\hat{e}$ can be efficiently computed.

Joux's Key Agreement Protocol

- A straightforward application of pairings is Joux's three-party one-round key agreement protocol.
- Note that the protocol is not interesting from a practical point of view.
- It is resistant to passive attacks; it needs at least an additional round for active attacks.

Three-party One-round Key Agreement (1)



Three-party One-round Key Agreement (2)

- Alice randomly selects a secret integer $a \in [1, n-1]$ and broadcasts point $[a]P$ to the other two parties.
- At the same time, Bob and Chris perform the same steps, broadcasting points $[b]P$ and $[c]P$.
- After receiving bP and cP, Alice (and also Bob and Chris) can compute the shared secret $K = \hat{e}([b]P, [c]P)^a = \hat{e}(P, P)^{abc}$.
- The system's security relies on the *Bilinear Diffie-Hellman Problem (BHDP)*: Given P, $[a]P$, $[b]P$, $[c]P$, compute $\hat{e}(P, P)^{abc}$.

Bilinear Mappings Implementation

- As with elliptic curves, cryptographers make specific recommendations on the bilinear mappings that should be used.
- In our case, the recommended way to go is the Ate pairing over a subclass of Barreto-Naehrig elliptic curves.

Speeding Up

The implementation uses C to take care of the cryptographic operations, but is about 100 slower than it would be if we were using *only* C. Reasons include:

- We spend a lot of time moving from Python to C and back, instead of *staying* in C.
- We do not vectorize operations.
- We do not use optimized mathematical operations such as windowed exponentation.

Vectorization Candidate

```python
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
```

And to Start in the First Place...

- The shuffle scheme we have described uses the *Common Reference String* (CRS) model.
- According to this model, during shuffling all parties have access to the CRS.
- The CRS must be generated in a way that it is shared among partners in the protocol.
- In other words, it must be calculated in a *Secure Multiparty Computation* fashion.

Secure Multiparty Multiplication (1)

- Suppose we have a set of n participants p_m, $1 \leq m \leq n$.
- We have two values s and t.
- We have $s = \sum_{i=1}^{k} s_i$ and $t = \sum_{j=1}^{k} t_j$.
- The different s_is and t_js are partitioned among the participants so that each participant has a subset of s_is and t_js and each s_i and t_j goes to one and only one participant.
- We call U_m the set of tuples (s_i, t_j) that goes to participant p_m.
- How can we compute $s \cdot t$ in a shared fashion? At the end we want the result to be shared among participants, so that we can get it only by bringing them all together.

Secure Multiparty Multiplication (2)

1. Each participant p_m computes $v_m = \sum_{(i,j)\in U_m} s_i t_j$.
2. Each participant sends v_m to all other participants.
3. Each participant adds locally all values received by other participants.

It is easy to see that $s \cdot t = \sum_{m=1}^{n} v_m$.

For more details, see Ueli Maurer, *Secure multi-party computation made simple*, Discrete Applied Mathematics, 154(2), 1 February 2006, pages 370–381.