# Optimized ZK Proofs for Paillier-Based 2PC ECDSA

## And Applications to Embedded Cryptocurrency Wallets

**Michael ADJEDJ**[*], Constantin BLOKH[*], Geoffroy COUTEAU[+],

Arik GALANSKY[*], Antoine JOUX[#], Nikolaos MAKRIYANNIS[*]

[*] : Fireblocks
[+] : CNRS, IRIF, Université Paris Cité
[#] : CISPA Helmholtz Center for Information Security

https://ia.cr/2024/1950

# Agenda

## 01
## Introduction

4 min

## 02
## 2PC ECDSA

12 min

## 03
## Additional Techniques

8 min

## 04
## Open Question & Future Work

< 1 min

# Agenda

01
**Introduction**

02
**2PC ECDSA**

03
**Additional Techniques**

04
**Open Question & Future Work**

# Motivations

- Trillions of dollars were transferred on blockchains
  - Transactions are signed mainly using ECDSA
  - 6T$ secured by Fireblocks MPC-ECDSA

# Motivations

- Trillions of dollars were transferred on blockchains
  - Transactions are signed mainly using ECDSA
  - 6T$ secured by Fireblocks MPC-ECDSA
- Target of choice for hackers
  - e.g. $1.5B stolen on Ethereum in February

# Motivations

- Trillions of dollars were transferred on blockchains
  - Transactions are signed mainly using ECDSA
  - 6T$ secured by Fireblocks MPC-ECDSA

- Target of choice for hackers
  - e.g. $1.5B stolen on Ethereum in February

- Millions of transactions per day

# Motivations

- Trillions of dollars were transferred on blockchains
  - Transactions are signed mainly using ECDSA
  - 6T$ secured by Fireblocks MPC-ECDSA

- Target of choice for hackers
  - e.g. $1.5B stolen on Ethereum in February

- Millions of transactions per day

- Transactions need to be highly secure, and avoid single-point-of-failure

# Motivations

- Trillions of dollars were transferred on blockchains
  - Transactions are signed mainly using ECDSA
  - 6T$ secured by Fireblocks MPC-ECDSA
- Target of choice for hackers
  - e.g. $1.5B stolen on Ethereum in February
- Millions of transactions per day
- Transactions need to be highly secure, and avoid single-point-of-failure
  - Threshold ECDSA seems to be the way to go

# The ideal solution for blockchain wallets

- Low computational overhead

# The ideal solution for blockchain wallets

- Low computational overhead

- Low communication overhead
  - Ideally, exchanged messages should be small enough

# The ideal solution for blockchain wallets

- Low computational overhead

- Low communication overhead
  - Ideally, exchanged messages should be small enough

- Optimal round complexity
  - Each round of communication induces additional latency
  - ~35 ms in the same region (e.g. Europe), ~150ms cross-region

# The ideal solution for blockchain wallets

- Low computational overhead

- Low communication overhead
  - Ideally, exchanged messages should be small enough

- Optimal round complexity
  - Each round of communication induces additional latency
  - ~35 ms in the same region (e.g. Europe), ~150ms cross-region

- Concurrent security
  - Parties should be able to handle millions of signatures in parallel securely

# Best-in-Class 2PC ECDSA

| | OLEs | Rounds | Comm. (KB) | Run time (ms) | Concurrent security |
|---|---|---|---|---|---|
| [Lin17] | 1 | 4 | 0.9 | 12 | ✗ |
| [DKLs18] (Ver. 2018) | 3 | 2 | 135 | 28 | ✓ |
| [XAXYC21] (Paillier) | 1 | 3 | $6.3^{\dagger}$ | $226^{\dagger}$ | ✓ |
| [XALCCXYZ23] | 1 | 3 | $4.1^{\dagger}$ | $209^{\dagger}$ | ✓ |
| [DKLs24] | 2 | 3 | 115 | 29 | ✓ |
| [BHL24] | $1^{\ddagger}$ | 2 | $5.6^{\ddagger}$ | $144^{\ddagger}$ | ✓ |

Benchmarks were run on an Intel(R) Core(TM) i7-1365U CPU, 1 thread

# Best-in-Class 2PC ECDSA

| | OLEs | Rounds | Comm. (KB) | Run time (ms) | Concurrent security |
|---|---|---|---|---|---|
| [Lin17] | 1 | 4 | 0.9 | 12 | ✗ |
| [DKLs18] (Ver. 2018) | 3 | 2 | 135 | 28 | ✓ |
| [XAXYC21] (Paillier) | 1 | 3 | 6.3† | 226† | ✓ |
| [XALCCXYZ23] | 1 | 3 | 4.1† | 209† | ✓ |
| [DKLs24] | 2 | 3 | 115 | 29 | ✓ |
| [BHL24] | 1‡ | 2 | 5.6‡ | 144‡ | ✓ |
| This Work | 1 | 2 | 2 | 48 | ✓ |

Benchmarks were run on an Intel(R) Core(TM) i7-1365U CPU, 1 thread

# Disclaimer


The talk will **not** focus **only** on ZK Proofs

The talk **will** focus on special optimizations

# Agenda

01
Introduction

02
2PC ECDSA

03
Additional Techniques

04
Open Question & Future Work

# ECDSA

# ECDSA

**Setup:** $(\mathbb{G}, g, q)$
**Key Generation:**

- *Private Key:* $x \leftarrow \mathbb{Z}_q$
- *Public key:* $X = g^x \in \mathbb{G}$

# ECDSA

**Setup:** $(\mathbb{G}, g, q)$
**Key Generation:**

- *Private Key*: $x \leftarrow \mathbb{Z}_q$
- *Public key*: $X = g^x \in \mathbb{G}$

**Sign:**

- *Input:* message $m$, Private Key: $x$
- $k \in \mathbb{Z}_q, \quad R = g^k, \quad r = R|_x$
- $\sigma = (\mathsf{H}(m) + x \cdot r) \cdot k^{-1} \bmod q$
- $(r, \sigma)$

# ECDSA

**Setup:** $(\mathbb{G}, g, q)$
**Key Generation:**

- *Private Key*: $x \leftarrow \mathbb{Z}_q$
- *Public key*: $X = g^x \in \mathbb{G}$

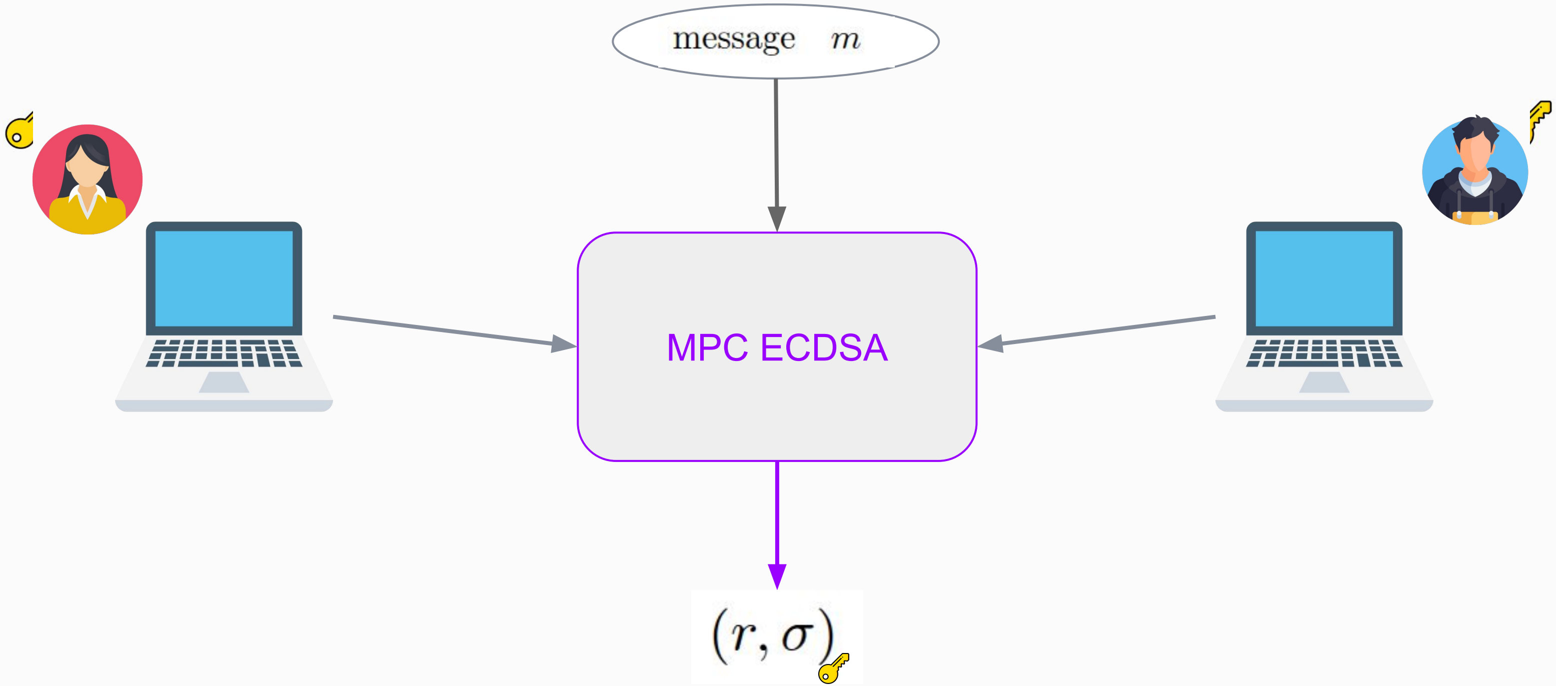
**Sign:**

- *Input:* message $m$, Private Key: $x$
- $k \in \mathbb{Z}_q, \quad R = g^k, \quad r = R|_x$
- $\sigma = (\mathsf{H}(m) + x \cdot r) \cdot k^{-1} \bmod q$
- $(r, \sigma)$

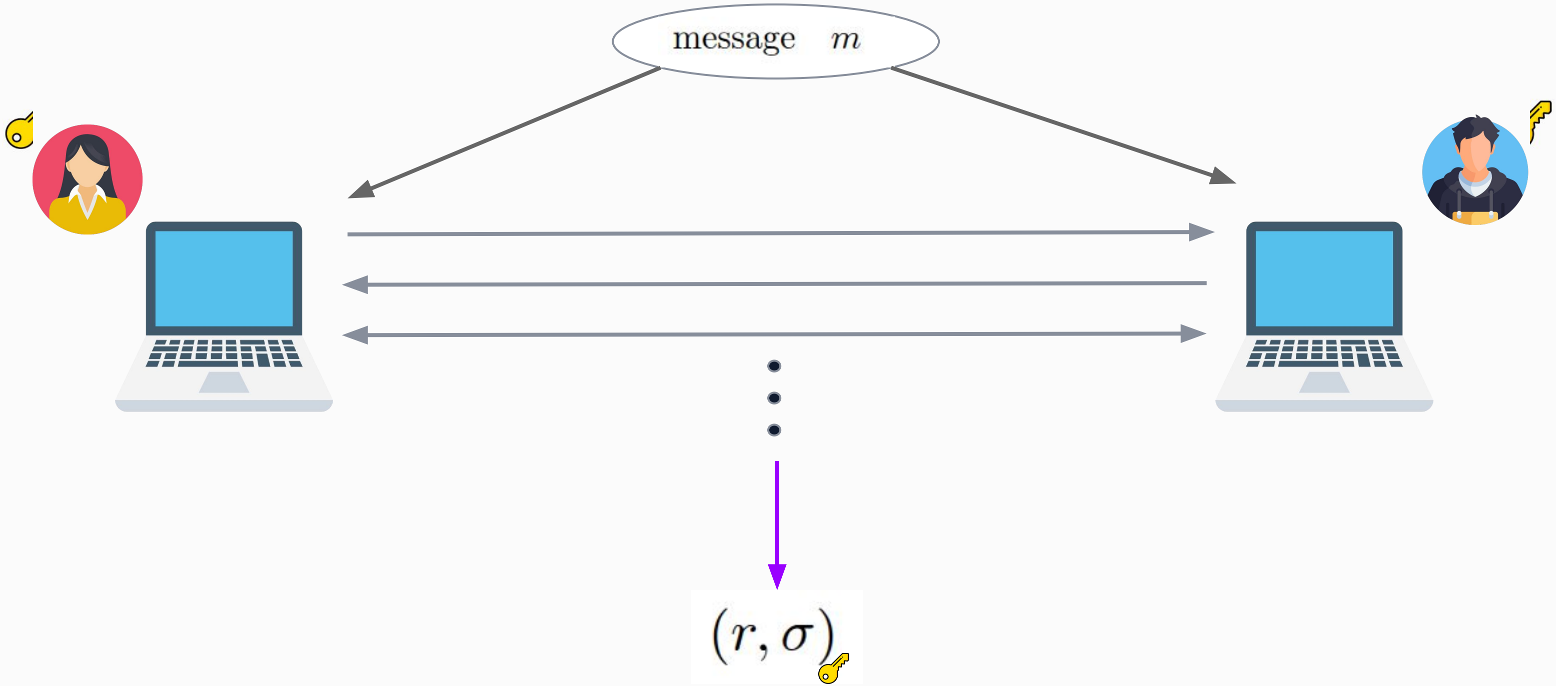
**Verify:**

- message $m$, signature $(r, \sigma)$
- $\left( X^{\frac{r}{s}} \cdot g^{\frac{\mathsf{H}(m)}{s}} \right)\Big|_x \stackrel{?}{=} r$

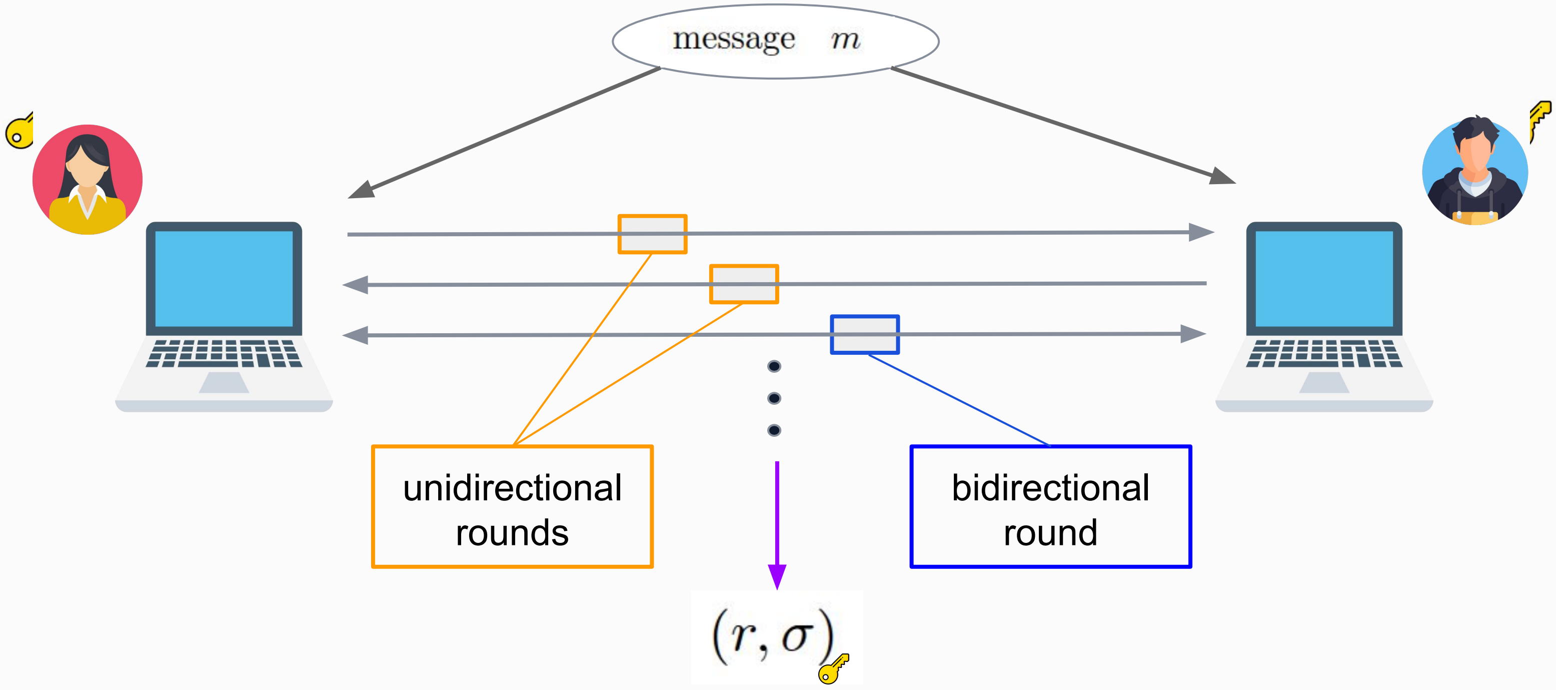
# 2-Party ECDSA



# 2-Party ECDSA



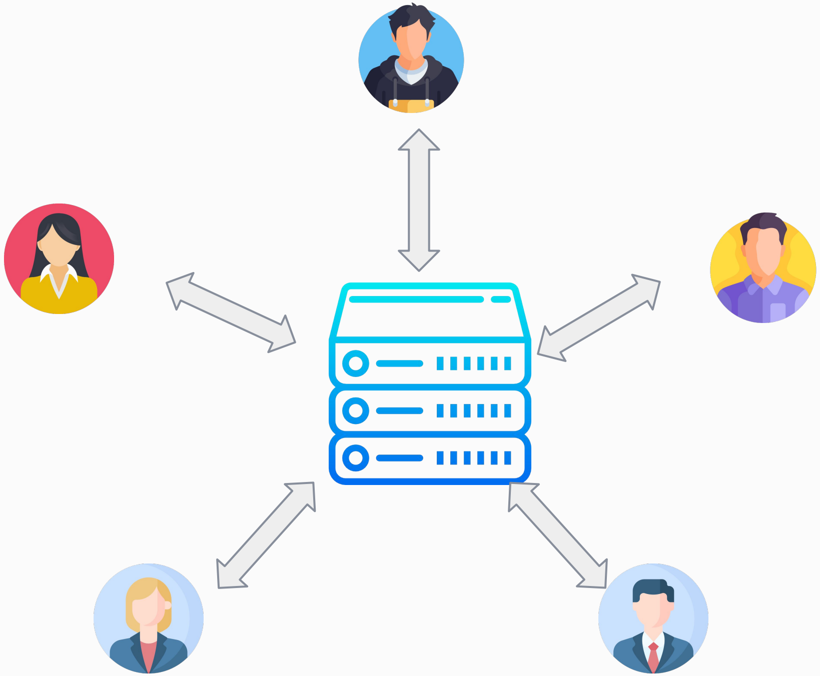
# 2-Party ECDSA



# Our 2-Party ECDSA

Preferred Setting:
Star-shaped topology

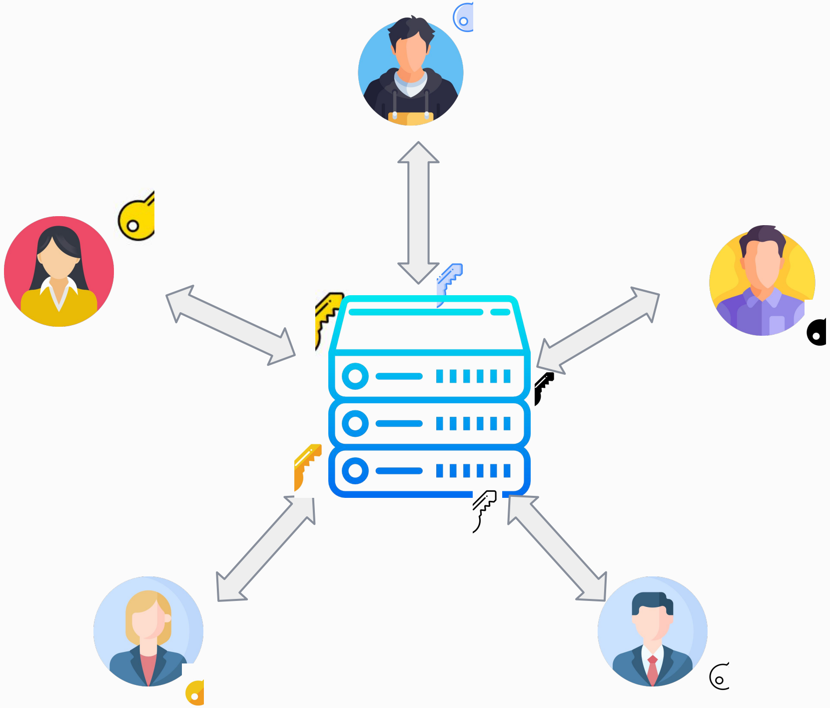
E.g. cryptocurrency wallet provider



# Our 2-Party ECDSA

Preferred Setting:
Star-shaped topology

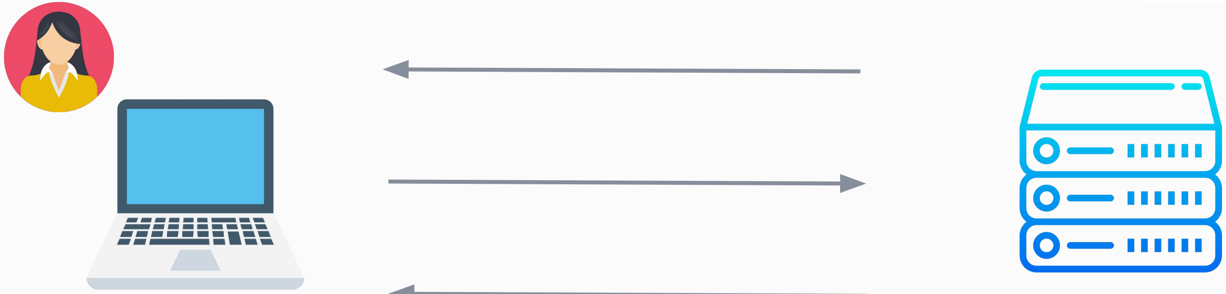
E.g. cryptocurrency wallet provider



# Our 2-Party ECDSA: Key Generation

$X_1 = g^{x_1}$

$X_2 = g^{x_2}$



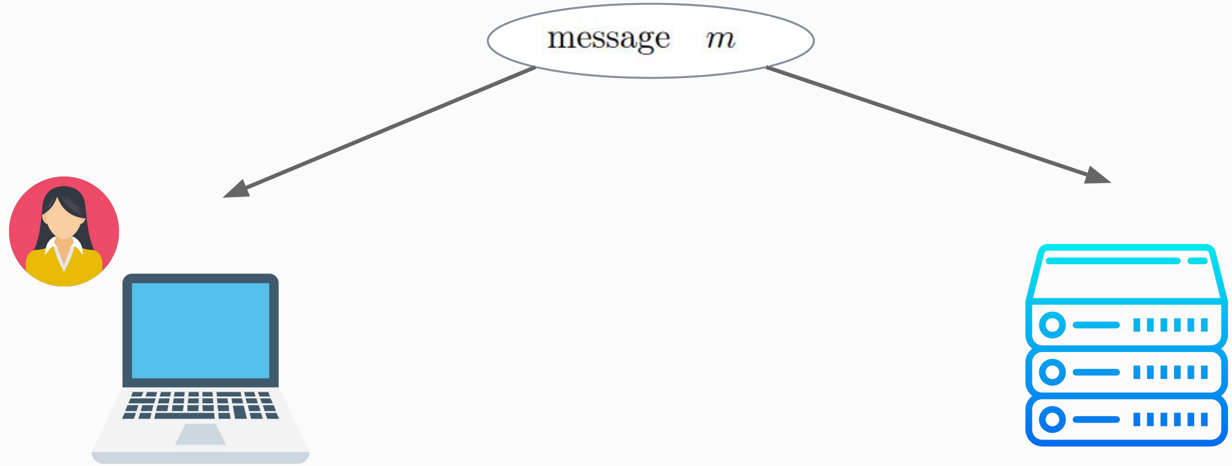
Common Public Key: $X = X_1 \cdot X_2$

Common Private Key: $x = x_1 + x_2$

# Our 2-Party ECDSA: Signature Generation

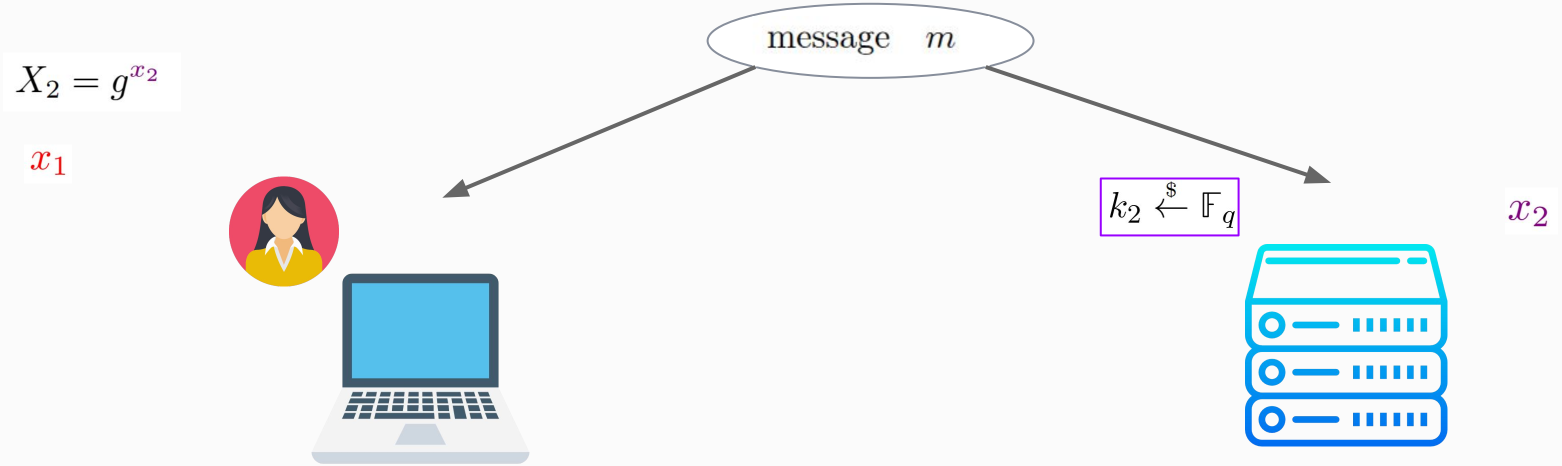
$X_2 = g^{x_2}$

$x_1$




$x_2$

# Our 2-Party ECDSA: Signature Generation



# Our 2-Party ECDSA: Signature Generation

$X_2 = g^{x_2}$

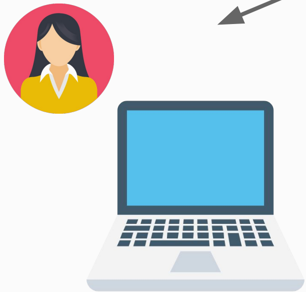
$x_1$

message $m$

$R_2 \leftarrow g^{k_2}$ and $Y \leftarrow X_1^{k_2}$

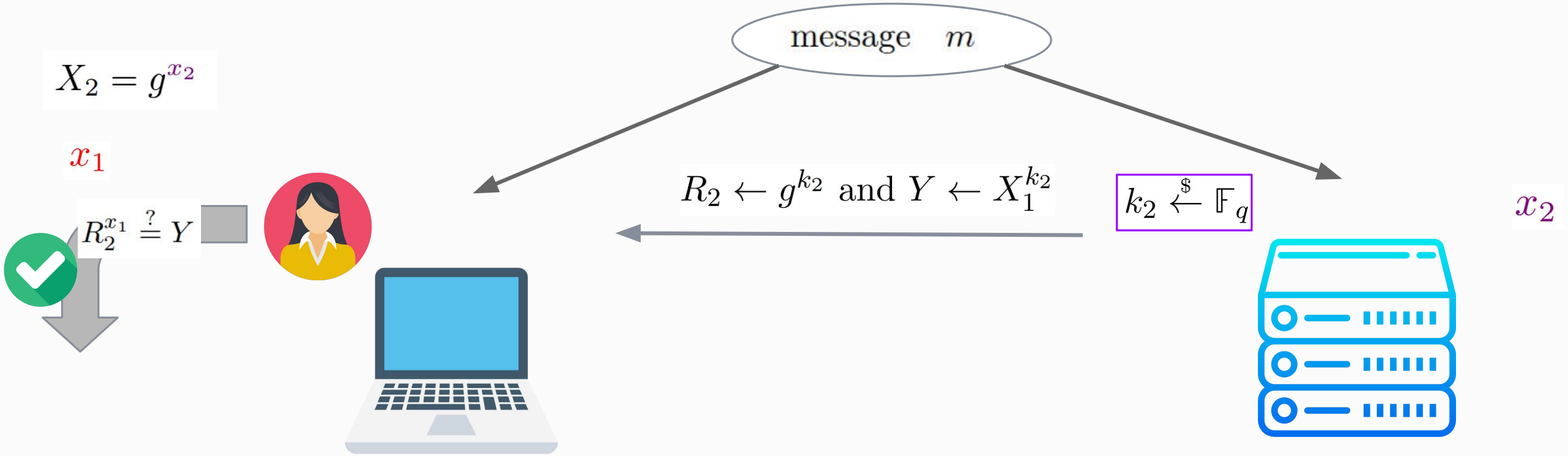
$\boxed{k_2 \xleftarrow{\$} \mathbb{F}_q}$

$x_2$

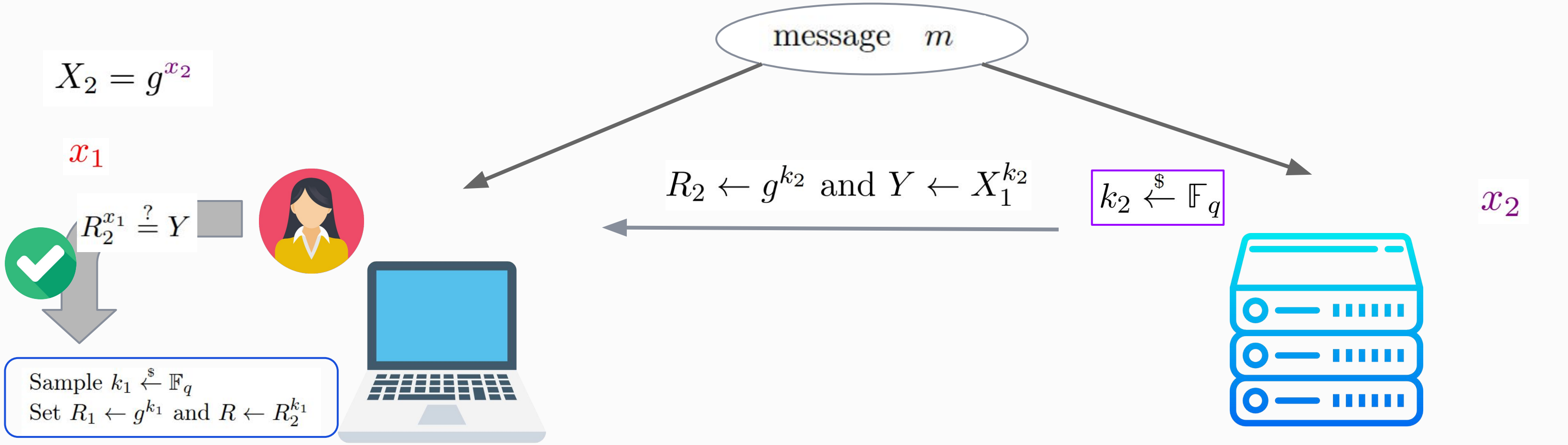


Fireblocks

# Our 2-Party ECDSA: Signature Generation



# Our 2-Party ECDSA: Signature Generation



# Our 2-Party ECDSA: Signature Generation

message $m$

$X_2 = g^{x_2}$

$x_1$

$R_2^{x_1} \stackrel{?}{=} Y$

Sample $k_1 \xleftarrow{\$} \mathbb{F}_q$
Set $R_1 \leftarrow g^{k_1}$ and $R \leftarrow R_2^{k_1}$

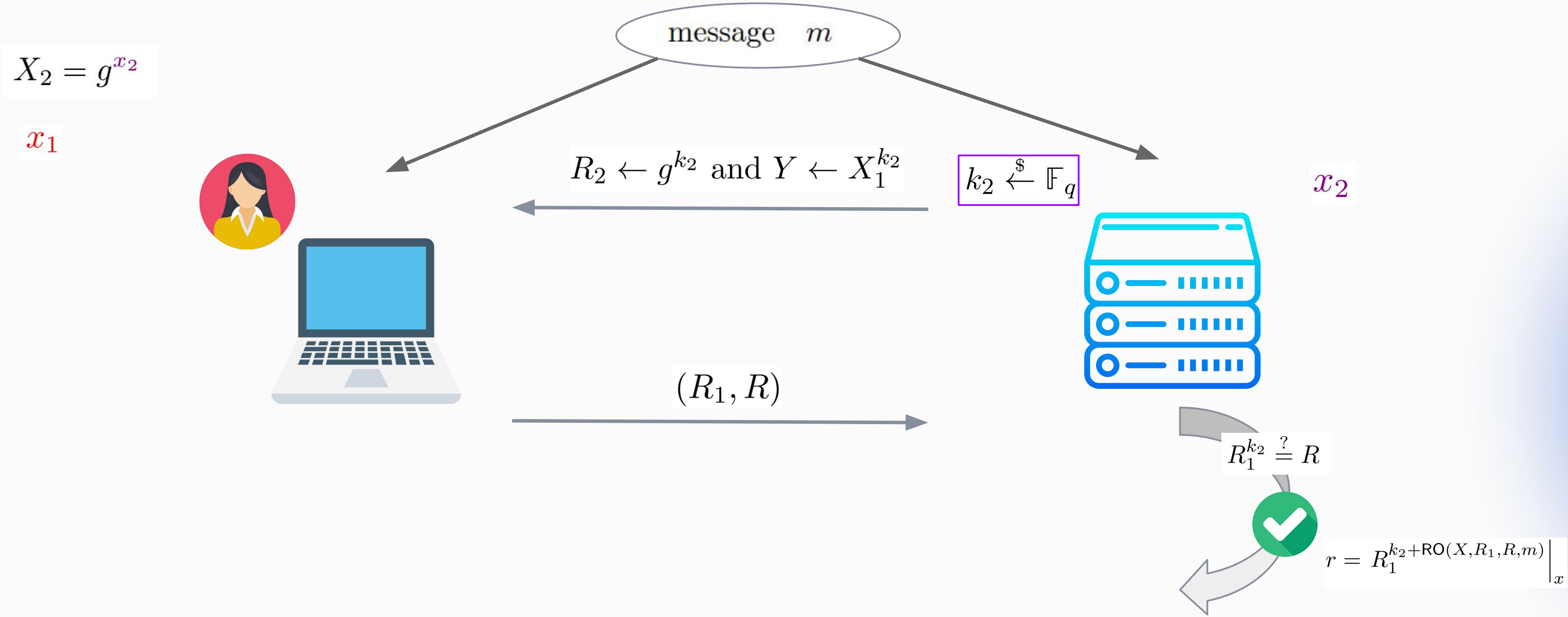
$R_2 \leftarrow g^{k_2}$ and $Y \leftarrow X_1^{k_2}$

$\boxed{k_2 \xleftarrow{\$} \mathbb{F}_q}$

$x_2$

$(R_1, R)$

# Our 2-Party ECDSA: Signature Generation



# Our 2-Party ECDSA: Signature Generation

message $m$

$X_2 = g^{x_2}$

$x_1$

$R_2 \leftarrow g^{k_2}$ and $Y \leftarrow X_1^{k_2}$

$\boxed{k_2 \xleftarrow{\$} \mathbb{F}_q}$

$x_2$

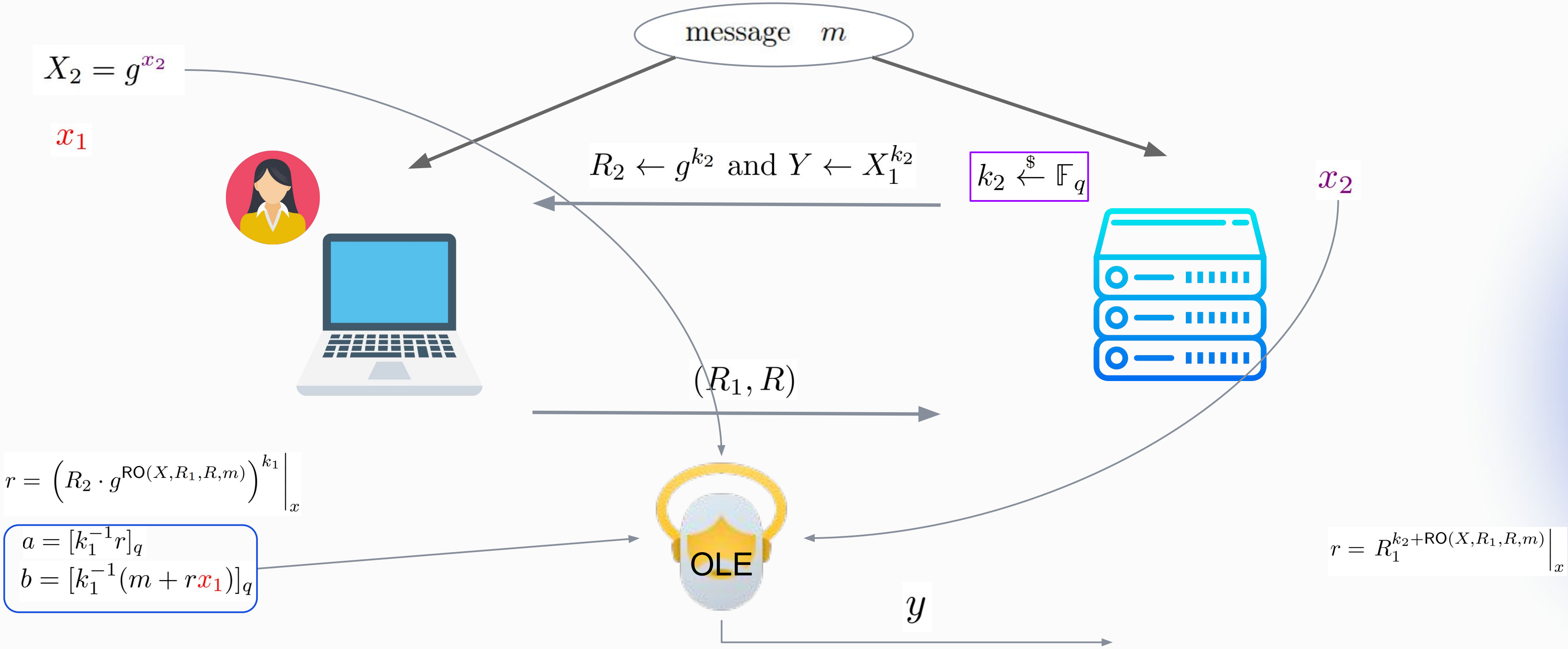
$(R_1, R)$

$r = \left(R_2 \cdot g^{\mathsf{RO}(X, R_1, R, m)}\right)^{k_1}\Big|_x$

$$\boxed{\begin{aligned} a &= [k_1^{-1} r]_q \\ b &= [k_1^{-1}(m + r x_1)]_q \end{aligned}}$$

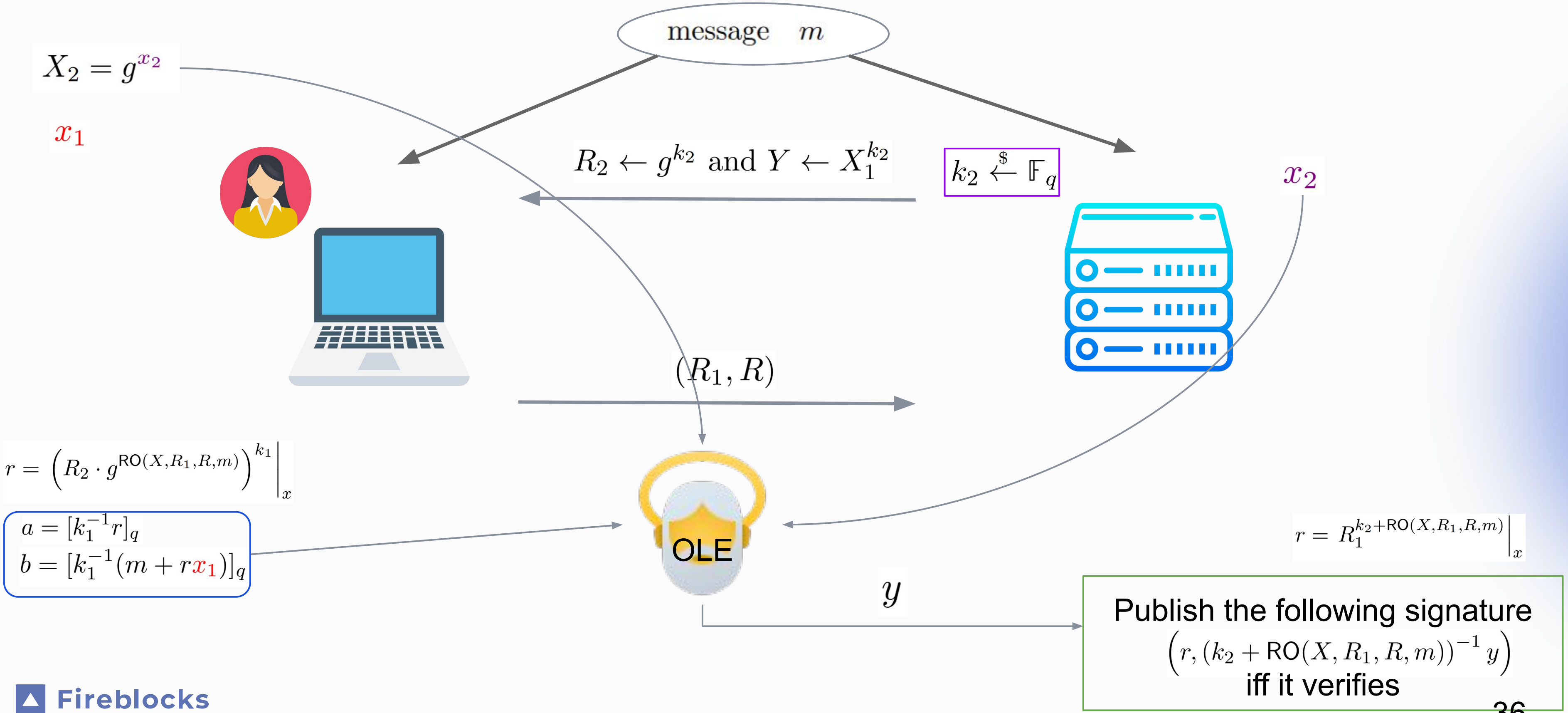
$r = R_1^{k_2 + \mathsf{RO}(X, R_1, R, m)}\Big|_x$

# Our 2-Party ECDSA: Signature Generation




Fireblocks

# Our 2-Party ECDSA: Signature Generation



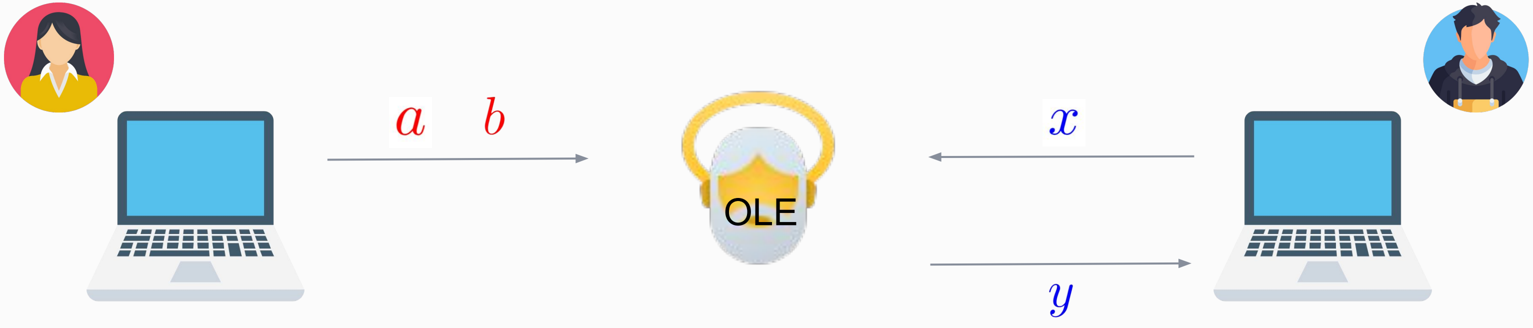

Fireblocks

# Oblivious Linear Evaluation (a.k.a. OLE)

# Oblivious Linear Evaluation (a.k.a. OLE)

# Oblivious Linear Evaluation (a.k.a. OLE)

# Oblivious Linear Evaluation (a.k.a. OLE)



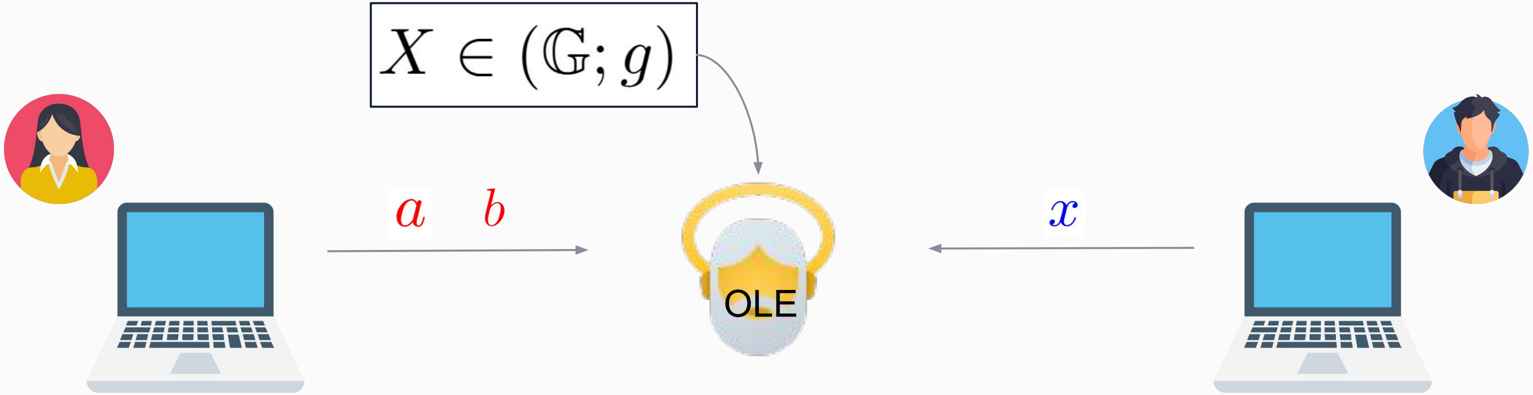

$$y = a \cdot x + b$$

# Oblivious Linear Evaluation (a.k.a. OLE)


$$y = a \cdot x + b$$

No one learns anything about the secrets of the other!

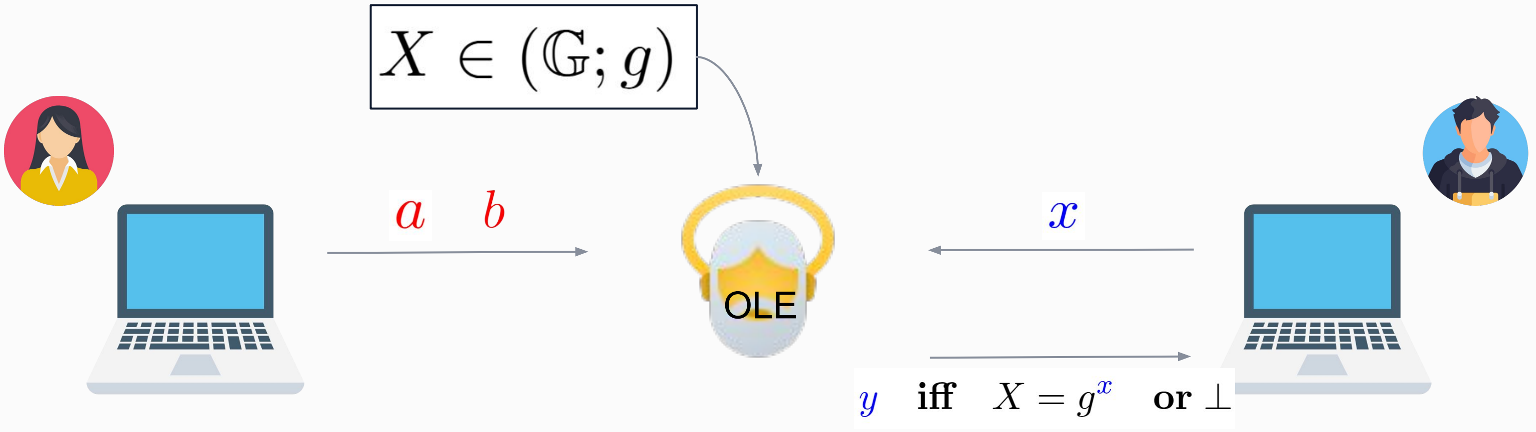
# Oblivious Linear Evaluation (a.k.a. OLE) modified




$$y = a \cdot x + b$$

No one learns anything about the secrets of the other!

# Oblivious Linear Evaluation (a.k.a. OLE) modified




$$y = a \cdot x + b$$

No one learns anything about the secrets of the other!

# OLE from Additively Homomorphic Encryption

- $\mathcal{E}$ Additively homomorphic encryption scheme

# OLE from Additively Homomorphic Encryption

- $\mathcal{E}$ Additively homomorphic encryption scheme
- Additively Homomorphic
  - $\mathcal{E}(m_1 + m_2) \overset{\sim}{=} \mathcal{E}(m_1) \oplus \mathcal{E}(m_2)$

# OLE from Additively Homomorphic Encryption

- $\mathcal{E}$ Additively homomorphic encryption scheme
- Additively Homomorphic
  - $\mathcal{E}(m_1 + m_2) \stackrel{\sim}{=} \mathcal{E}(m_1) \oplus \mathcal{E}(m_2)$
- Multiplication by a constant
  - $\mathcal{E}(c \cdot m_1) \stackrel{\sim}{=} c \odot \mathcal{E}(m_1)$

# OLE from Additively Homomorphic Encryption

- $\mathcal{E}$ Additively homomorphic encryption scheme
- Additively Homomorphic
  - $\mathcal{E}(m_1 + m_2) \stackrel{\sim}{=} \mathcal{E}(m_1) \oplus \mathcal{E}(m_2)$
- Multiplication by a constant
  - $\mathcal{E}(c \cdot m_1) \stackrel{\sim}{=} c \odot \mathcal{E}(m_1)$
- Instantiation of OLE
  - $a \odot \mathcal{E}(x_2) \oplus b \stackrel{\sim}{=} \mathcal{E}(a \cdot x_2 + b)$

# Paillier Encryption

**KeyGen:**

- $(p, q)$ prime numbers, $N = pq$
- Public: $(N)$ Private: $(p, q, \varphi(N))$

# Paillier Encryption

**KeyGen:**

- $(p, q)$ prime numbers, $N = pq$
- Public: $(N)$  Private: $(p, q, \varphi(N))$

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output

$$C = \mathrm{enc}_N(m; \rho) = (1 + mN) \cdot \rho^N \mod N^2.$$

# Paillier Encryption

**KeyGen:**

- $(p, q)$ prime numbers, $N = pq$
- Public: $(N)$ Private: $(p, q, \varphi(N))$

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output

$$C = \mathrm{enc}_N(m;\rho) = (1 + mN) \cdot \rho^N \mod N^2.$$

**Decrypt:**

- On input $C \in \mathbb{Z}_{N^2}^*$ and $\varphi(N)$, output $m = \left(\frac{[C^{\varphi(N)}]_{N^2} - 1}{N}\right) \cdot \varphi(N)^{-1} \mod N$

# Paillier Encryption

**KeyGen:**

- $(p, q)$ prime numbers, $N = pq$
- Public: $(N)$ Private: $(p, q, \varphi(N))$

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output

$$C = \mathrm{enc}_N(m;\rho) = (1 + mN) \cdot \rho^N \mod N^2.$$

**Decrypt:**

- On input $C \in \mathbb{Z}_{N^2}^*$ and $\varphi(N)$, output $m = \left(\frac{[C^{\varphi(N)}]_{N^2}-1}{N}\right) \cdot \varphi(N)^{-1} \mod N$

Additional ZK proofs (adapted from [CGGMP20]):

- $\Pi_{mod}$: N is the product of 2 primes congruent to *3 mod 4*, and gcd(N, φ(N)) = 1
- $\Pi_{fac}$: N has no factor larger than $c \cdot \sqrt{N}$ with *c* "small" ($\sim 2^{l+l/2}$)

[CGGMP20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts". In: ACM CCS 2020

# Instantiation of OLE with Paillier Encryption

- Comes from partial homomorphic property of Paillier's encryption:
  - Additively homomorphic
    - $\mathsf{Enc}_N(m_1) \times \mathsf{Enc}_N(m_2) \overset{\sim}{=} \mathsf{Enc}_N(m_1 + m_2)$
  - External product
    - $\mathsf{Enc}_N(m_1 \times m_2) \overset{\sim}{=} \mathsf{Enc}_N(m_1)^{m2}$

# Instantiation of OLE with Paillier Encryption

- Comes from partial homomorphic property of Paillier's encryption:
    - Additively homomorphic
        - $\mathsf{Enc}_N(m_1) \times \mathsf{Enc}_N(m_2) \stackrel{\sim}{=} \mathsf{Enc}_N(m_1 + m_2)$
    - External product
        - $\mathsf{Enc}_N(m_1 \times m_2) \stackrel{\sim}{=} \mathsf{Enc}_N(m_1)^{m2}$

- Requires ZK Proofs $\Pi_{\text{fac}}$ and $\Pi_{\text{mod}}$

# Instantiation of OLE with Paillier Encryption

- Comes from partial homomorphic property of Paillier's encryption:
  - Additively homomorphic
    - $\mathsf{Enc}_N(m_1) \times \mathsf{Enc}_N(m_2) \overset{\sim}{=} \mathsf{Enc}_N(m_1 + m_2)$
  - External product
    - $\mathsf{Enc}_N(m_1 \times m_2) \overset{\sim}{=} \mathsf{Enc}_N(m_1)^{m2}$

- Requires ZK Proofs $\Pi_{\text{fac}}$ and $\Pi_{\text{mod}}$

- The modified OLE (with public share) requires a PoK of discrete logarithm
  - Done during key generation

# $\Pi_{mod}$ : N is a bi-prime and gcd(N, φ(N)) = 1

P

V

$y$

Determine $(a, b)$ $s.t.$

$y_0 = (-1)^a \cdot 2^{Nb} \cdot y \in \mathsf{QR}_N$

$y \xleftarrow{\$} \mathbb{Z}_N^*$

$x = \sqrt[4]{y_0} \mod N$

$z = \sqrt[N]{y} \mod N$

$(a, b, x, z)$

$$\begin{cases} N \stackrel{?}{=} 1 \mod 4 \\ N \text{ is a composite number} \\ x^4 \stackrel{?}{=} (-1)^a \cdot 2^{Nb} \cdot y \mod N \in \mathbb{Z}_N^* \\ y \stackrel{?}{=} z^N \mod N \in \mathbb{Z}_N^* \end{cases}$$

# How to deal with dishonest/malicious parties?

- Need Zero-Knowledge proofs
  - Instantiation of OLE
  - Key Generation

# How to deal with dishonest/malicious parties?

- Need Zero-Knowledge proofs
  - Instantiation of OLE
  - Key Generation

- Common public nonce ($R$) shifted with RO output
  - Deal with dishonest nonce generation by the client

# How to deal with dishonest/malicious parties?

- Need Zero-Knowledge proofs
  - Instantiation of OLE
  - Key Generation

- Common public nonce (*R*) shifted with RO output
  - Deal with dishonest nonce generation by the client

- Need integer commitment scheme
  - Damgård-Fujisaki (2-)commitment scheme

# How to deal with dishonest/malicious parties?

- Need Zero-Knowledge proofs
  - Instantiation of OLE
  - Key Generation

- Common public nonce (*R*) shifted with RO output
  - Deal with dishonest nonce generation by the client

- Need integer commitment scheme
  - Damgård-Fujisaki (2-)commitment scheme
  - Both Client & Server
    - Also need to prove that parameters are well-formed

# Damgård-Fujisaki (2-)Commitments

**Setup:**

- $(p, q)$ strong prime numbers, $N = pq$
- $v \in \mathtt{QR}(N)$, $\lambda_1, \lambda_2 \in \mathbb{Z}_{\varphi(N)}$
- $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$
- Public: $(N, u_1, u_2)$ Private: $(p, q, \lambda_1, \lambda_2)$

**Parameters well-formedness proof:**

*Secret Input.* P holds $\lambda_1, \lambda_2 \in [\varphi(N)]$ such that $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$.

1. P sends $A \leftarrow v^{\alpha} \mod N$ for $\alpha \xleftarrow{\$} [\varphi(N)]$.
2. V replies with random challenges $e_1 \xleftarrow{\$} \{0,1\}$, $e_2 \xleftarrow{\$} \{0,1\}$.
3. P returns $z \leftarrow \alpha + e_1\lambda_1 + e_2\lambda_2 \mod \varphi(N)$.

*Verification.* V accepts if $v^z = A \cdot u_1^{e_1} \cdot u_2^{e_2} \mod N$

**Commitment:**

For $m_1, m_2 \in \mathbb{Z}$ and $(N, v, u_1, u_2)$, sample $\rho \leftarrow [N \cdot 2^{\ell}]$ and output

$$C = {u_1}^{m_1} \cdot {u_2}^{m_2} \cdot v^{\rho} \mod N.$$

: Computationally intensive : Bandwidth consuming

# Agenda

01
Introduction

02
2PC ECDSA

03
**Additional Techniques**

04
Open Question & Future Work

# A new class of prime numbers: Tough Primes

# A new class of prime numbers: Tough Primes

- Strong prime
  - $p = 2 \cdot p_0 + 1$ with $p_0$ prime

# A new class of prime numbers: Tough Primes

- Strong prime
    - $p = 2 \cdot p_0 + 1$ with $p_0$ prime

- Tough prime
    - $l$ security parameter

# A new class of prime numbers: Tough Primes

- Strong prime
  - $p = 2 \cdot p_0 + 1$ with $p_0$ prime
- Tough prime
  - $l$ security parameter
  - $p = 2 \cdot p_0 \cdot \ldots \cdot p_t + 1$ where $p_i$ at least $2l$-bits primes
    - if $l = 128, |p_i| = 256$

# A new class of prime numbers: Tough Primes

- Strong prime
  - $p = 2 \cdot p_0 + 1$ with $p_0$ prime
- Tough prime
  - $l$ security parameter
  - $p = 2 \cdot p_0 \cdot \ldots \cdot p_t + 1$ where $p_i$ at least 2$l$-bits primes
    - if $l = 128, |p_i| = 256$
- To the best of our knowledge, it's the first time this is proposed

# A new class of prime numbers: Tough Primes

- Strong prime
  - $p = 2 \cdot p_0 + 1$ with $p_0$ prime
- Tough prime
  - $l$ security parameter
  - $p = 2 \cdot p_0 \cdot \ldots \cdot p_t + 1$ where $p_i$ at least $2l$-bits primes
    - if $l = 128, |p_i| = 256$
- To the best of our knowledge, it's the first time this is proposed

Claim: Tough primes provide **identical security guarantees**, compared to strong primes, in a variety of contexts e.g. Damgård-Fujisaki commitment, ZK proofs, …

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

| Bit size \ Prime type | Regular | | Tough | | Strong | |
|---|---|---|---|---|---|---|
| | average | std dev. | average | std dev. | average | std dev. |
| 1024 | 23.2 | 12.6 | 37.1 | 18.4 | 586.7 | 555.8 |
| 1536 | 70.5 | 35.9 | 98.9 | 53.9 | $3.6\times10^3$ | $4.2\times10^3$ |
| 2048 | 204.7 | 137.1 | 235.5 | 143.1 | $14.8\times10^3$ | $15.3\times10^3$ |

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

| Prime type / Bit size | Regular average | Regular std dev. | Tough average | Tough std dev. | Strong average | Strong std dev. |
|---|---|---|---|---|---|---|
| 1024 | 23.2 | 12.6 | 37.1 | 18.4 | 586.7 | 555.8 |
| 1536 | $\boxed{70.5}$ | 35.9 | $\boxed{98.9}$ | 53.9 | $3.6\times10^3$ | $4.2\times10^3$ |
| 2048 | 204.7 | 137.1 | 235.5 | 143.1 | $14.8\times10^3$ | $15.3\times10^3$ |

x1.4

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

| Prime type / Bit size | Regular | | Tough | | Strong | |
|---|---|---|---|---|---|---|
| | average | std dev. | average | std dev. | average | std dev. |
| 1024 | 23.2 | 12.6 | 37.1 | 18.4 | 586.7 | 555.8 |
| 1536 | $\boxed{70.5}$ | 35.9 | 98.9 | 53.9 | $\boxed{3.6\times10^3}$ | $4.2\times10^3$ |
| 2048 | 204.7 | 137.1 | 235.5 | 143.1 | $14.8\times10^3$ | $15.3\times10^3$ |

x50

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

| Bit size \ Prime type | Regular average | Regular std dev. | Tough average | Tough std dev. | Strong average | Strong std dev. |
|---|---|---|---|---|---|---|
| 1024 | 23.2 | 12.6 | 37.1 | 18.4 | 586.7 | 555.8 |
| 1536 | 70.5 | 35.9 | 98.9 | 53.9 | $3.6\times10^3$ | $4.2\times10^3$ |
| 2048 | $\boxed{204.7}$ | 137.1 | $\boxed{235.5}$ | 143.1 | $14.8\times10^3$ | $15.3\times10^3$ |

x1.2

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

| Prime type / Bit size | Regular | | Tough | | Strong | |
|---|---|---|---|---|---|---|
| | average | std dev. | average | std dev. | average | std dev. |
| 1024 | 23.2 | 12.6 | 37.1 | 18.4 | 586.7 | 555.8 |
| 1536 | 70.5 | 35.9 | 98.9 | 53.9 | $3.6\times10^3$ | $4.2\times10^3$ |
| 2048 | $\boxed{204.7}$ | 137.1 | 235.5 | 143.1 | $\boxed{14.8\times10^3}$ | $15.3\times10^3$ |

x72

Fireblocks

# Use of Tough Primes in our Protocol

- Generation of Damgård-Fujisaki parameters
  - Server’s side during the setup
  - Client’s side during key generation

- ~15x improvement for both Client & Server

# Short Exponents (SE)

- SEDL: SE Discrete Logarithm

**Definition 3.4** (SEDL over $\mathbb{Z}_N^*$). Define SEDL such that $(N, t, [t^x]_N) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)$ for $(N; p, q) \xleftarrow{\$} \mathsf{SampleRSA}(1^\ell)$, $t \xleftarrow{\$} \mathsf{QR}_N$ and $x \xleftarrow{\$} \pm 2^{2\ell}$. We say that *small-exponent discrete-log (SEDL) holds true* if for every PPTM $\mathsf{A}$, there exists a negligible function $\mu : \mathbb{N} \to \mathbb{R}$ such that

$$\Pr_{(N,t,s) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)} \left[ x_0 \xleftarrow{\$} \mathsf{A}(1^\ell, N, t, s) \;\text{ s.t. }\; t^{x_0} = s \bmod N \right] \leq \mu(\ell).$$

# Short Exponents (SE)

- SEDL: SE Discrete Logarithm

**Definition 3.4** (SEDL over $\mathbb{Z}_N^*$). Define SEDL such that $(N, t, [t^x]_N) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)$ for $(N; p, q) \xleftarrow{\$}$ $\mathsf{SampleRSA}(1^\ell)$, $t \xleftarrow{\$} \mathsf{QR}_N$ and $x \xleftarrow{\$} \pm 2^{2\ell}$. We say that *small-exponent discrete-log (SEDL) holds true* if for every PPTM A, there exists a negligible function $\mu : \mathbb{N} \to \mathbb{R}$ such that

$$\Pr_{(N,t,s) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)} \left[ x_0 \xleftarrow{\$} \mathsf{A}(1^\ell, N, t, s) \text{ s.t. } t^{x_0} = s \bmod N \right] \leq \mu(\ell).$$

- SEI: SE Indistinguishability

**Definition 3.5** (SEI over $\mathbb{Z}_N^*$). Define distribution ensemble SEI such that $(N, t, [t^{\alpha x + (1-\alpha)y}]_N, \alpha) \xleftarrow{\$}$ $\mathsf{SEDL}(1^\ell)$ for $N \xleftarrow{\$} \mathsf{SampleRSA}(1^\ell)$, $t \xleftarrow{\$} \mathsf{QR}_N$, $x \xleftarrow{\$} \pm 2^{2\ell}$, $y \xleftarrow{\$} \pm N$ and $\alpha \xleftarrow{\$} \{0, 1\}$. We say that *small-exponent indistinguishability (SEI) holds true* if for every PPTM A, there exists a negligible function $\mu : \mathbb{N} \to \mathbb{R}$ such that

$$\Pr_{(N,t,s,\alpha) \xleftarrow{\$} \mathsf{SEI}(1^\ell)} \left[ \alpha_0 \xleftarrow{\$} \mathsf{A}(1^\ell, N, t, s) \text{ s.t. } \alpha_0 = \alpha \right] \leq \frac{1}{2} + \mu(\ell).$$

# Short Exponents (SE)

- SEDL: SE Discrete Logarithm

**Definition 3.4** (SEDL over $\mathbb{Z}_N^*$)**.** Define SEDL such that $(N, t, [t^x]_N) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)$ for $(N; p, q) \xleftarrow{\$} \mathsf{SampleRSA}(1^\ell)$, $t \xleftarrow{\$} \mathsf{QR}_N$ and $x \xleftarrow{\$} \pm 2^{2\ell}$. We say that *small-exponent discrete-log (SEDL) holds true* if for every PPTM A, there exists a negligible function $\mu : \mathbb{N} \to \mathbb{R}$ such that

$$\Pr_{(N,t,s) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)} \left[ x_0 \xleftarrow{\$} \mathsf{A}(1^\ell, N, t, s) \ \text{ s.t. } \ t^{x_0} = s \bmod N \right] \le \mu(\ell).$$

- SEI: SE Indistinguishability

**Definition 3.5** (SEI over $\mathbb{Z}_N^*$)**.** Define distribution ensemble SEI such that $(N, t, [t^{\alpha x + (1-\alpha) y}]_N, \alpha) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)$ for $N \xleftarrow{\$} \mathsf{SampleRSA}(1^\ell)$, $t \xleftarrow{\$} \mathsf{QR}_N$, $x \xleftarrow{\$} \pm 2^{2\ell}$, $y \xleftarrow{\$} \pm N$ and $\alpha \xleftarrow{\$} \{0, 1\}$. We say that *small-exponent indistinguishability (SEI) holds true* if for every PPTM A, there exists a negligible function $\mu : \mathbb{N} \to \mathbb{R}$ such that

$$\Pr_{(N,t,s,\alpha) \xleftarrow{\$} \mathsf{SEI}(1^\ell)} \left[ \alpha_0 \xleftarrow{\$} \mathsf{A}(1^\ell, N, t, s) \ \text{ s.t. } \ \alpha_0 = \alpha \right] \le \frac{1}{2} + \mu(\ell).$$

- Shown that SEI reduces to SEDL in prime order group [KK04]. We showed it also reduces in the case of unknown order groups (e.g. RSA multiplicative group)

[KK04] Takeshi Koshiba and Kaoru Kurosawa. “Short Exponent Diffie-Hellman Problems”. In: PKC 2004.

# Use of Short Exponents: Paillier Encryption

- Paillier encryption

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output
$$C = \mathrm{enc}_N(m; \rho) = (1 + mN) \cdot \rho^N \mod N^2.$$

# Use of Short Exponents: Paillier Encryption

- Paillier encryption

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output
$$C = \mathrm{enc}_N(m;\rho) = (1 + mN) \cdot \rho^N \mod N^2.$$

# Use of Short Exponents: Paillier Encryption

- Paillier encryption

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output

$$C = \mathrm{enc}_N(m; \rho) = (1 + mN) \cdot \rho^N \mod N^2.$$

  - Pick $\rho = \rho_0^{2N}$ => can be a constant to the Paillier parameters

# Use of Short Exponents: Paillier Encryption

- Paillier encryption

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output
$$C = \mathrm{enc}_N(m;\rho) = (1+mN)\cdot\rho^N \mod N^2.$$

- Pick $\rho = \rho_0^{2N}$ => can be a constant to the Paillier parameters
  - Needs to be checked by the clients

# Use of Short Exponents: Paillier Encryption

- Paillier encryption

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output
$$C = \mathrm{enc}_N(m;\rho) = (1+mN)\cdot \rho^N \mod N^2.$$

- Pick $\rho = \rho_0^{2N}$ => can be a constant to the Paillier parameters
  - Needs to be checked by the clients
- $C = \mathrm{enc}_N(m;\rho,r) = (1+mN)\cdot \rho^r \mod N^2 \text{ s.t. } r \xleftarrow{\$} [2^{2\ell}]$

# Use of Short Exponents: Paillier Encryption

- Paillier encryption

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output
$$C = \mathrm{enc}_N(m; \rho) = (1 + mN) \cdot \rho^N \mod N^2.$$

- Pick $\rho = \rho_0^{2N}$ => can be a constant to the Paillier parameters
  - Needs to be checked by the clients
- $C = \mathrm{enc}_N(m; \rho, r) = (1 + mN) \cdot \rho^r \mod N^2 \text{ s.t. } r \xleftarrow{\$} [2^{2\ell}]$
  - Computation time linear to exponent bitlength

# Use of Short Exponents: Paillier Encryption

- Paillier encryption

  **Encrypt:**

  - On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
  - Output
  $$C = \mathrm{enc}_N(m; \rho) = (1 + mN) \cdot \rho^N \mod N^2.$$

  - Pick $\rho = \rho_0^{2N}$ => can be a constant to the Paillier parameters
    - Needs to be checked by the clients
  - $C = \mathrm{enc}_N(m; \rho, r) = (1 + mN) \cdot \rho^r \mod N^2 \text{ s.t. } r \xleftarrow{\$} [2^{2\ell}]$
    - Computation time linear to exponent bitlength

- Decryption is still possible since

$$(\rho^r)^{\varphi(N)} = (\rho_0^{2N})^{\varphi(N)} = \left(\rho_0^{N \cdot \varphi(N)}\right)^2 = 1^2 = 1$$

Fireblocks

# Use of Short Exponents: Paillier Benchmarks (µs)

| Bitlength | Full-size exponents | Short exponents | Improvement Factor |
|---|---|---|---|
| 2048 | 4985 | 642 | 7.7x |
| 4096 | 38379 | 2512 | 15.2x |

l = 128

# Use of Short Exponents: Damgård-Fujisaki

**Setup:**

- $(p, q)$ strong prime numbers, $N = pq$
- $v \in \mathtt{QR}(N)$, $\lambda_1, \lambda_2 \in \mathbb{Z}_{\varphi(N)}$
- $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$
- Public: $(N, u_1, u_2)$ Private: $(p, q, \lambda_1, \lambda_2)$

**Parameters well-formedness proof:**

*Secret Input.* P holds $\lambda_1, \lambda_2 \in [\varphi(N)]$ such that $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$.

1. P sends $A \leftarrow v^{\alpha} \mod N$ for $\alpha \xleftarrow{\$} [\varphi(N)]$.
2. V replies with random challenges $e_1 \xleftarrow{\$} \{0,1\}, e_2 \xleftarrow{\$} \{0,1\}$.
3. P returns $z \leftarrow \alpha + e_1\lambda_1 + e_2\lambda_2 \mod \varphi(N)$.

*Verification.* V accepts if $v^z = A \cdot u_1^{e_1} \cdot u_2^{e_2} \mod N$

**Commitment:**

For $m_1, m_2 \in \mathbb{Z}$ and $(N, v, u_1, u_2)$, sample $\rho \leftarrow [N \cdot 2^{\ell}]$ and output

$$C = {u_1}^{m_1} \cdot {u_2}^{m_2} \cdot v^{\rho} \mod N.$$

: Computationally intensive : Bandwidth consuming

Fireblocks

# Agenda

01
Introduction

02
2PC ECDSA

03
Additional Techniques

04
Open Question & Future Work

# Work in Progress and Future Work

- [WIP] Protocol under deployment in real-world
    - Code / Protocol being audited
    - Open-source (after audit): https://github.com/fireblocks/mpc-lib

- Improve the protocol even further
    - Reduce computation/Communication

- Tough primes
    - Applications to other use cases

Fireblocks

Université Paris Cité

cnrs

CISPA

# Thank You

https://ia.cr/2024/1950

# The Best of Both Worlds: Round-Optimized 2PC ECDSA at the Cost of only 1 OLE

## And Applications to Embedded Cryptocurrency Wallets

**Michael ADJEDJ**[*], Constantin BLOKH[*], Geoffroy COUTEAU[+], Antoine JOUX[#], Nikolaos MAKRIYANNIS[*]

[*] : Fireblocks
[+] : CNRS, IRIF, Université Paris Cité
[#] : CISPA Helmholtz Center for Information Security

https://ia.cr/2024/1950

# Desirable Properties for a Threshold ECDSA

- Low computational overhead

- Low communication overhead
    - Ideally, exchanged messages should be small enough

- Optimal round complexity
    - Each round of communication induces additional latency

- Concurrent security
    - Parties should be able to handle millions of signatures in parallel securely

# Best-in-Class 2PC ECDSA

| | OLEs | Rounds | Comm. (KB) | Run time (ms) | Concurrent security |
|---|---|---|---|---|---|
| [Lin17] | 1 | 4 | 0.9 | 12 | ✗ |
| [DKLs18] (Ver. 2018) | 3 | 2 | 135 | 28 | ✓ |
| [XAXYC21] (Paillier) | 1 | 3 | 6.3† | 226† | ✓ |
| [XALCCXYZ23] | 1 | 3 | 4.1† | 209† | ✓ |
| [DKLs24] | 2 | 3 | 115 | 29 | ✓ |
| [BHL24] | 1‡ | 2 | 5.6‡ | 144‡ | ✓ |

Benchmarks were run on an Intel(R) Core(TM) i7-1365U CPU, 1 thread

# Best-in-Class 2PC ECDSA

| | OLEs | Rounds | Comm. (KB) | Run time (ms) | Concurrent security |
|---|---|---|---|---|---|
| [Lin17] | 1 | 4 | 0.9 | 12 | ✗ |
| [DKLs18] (Ver. 2018) | 3 | 2 | 135 | 28 | ✓ |
| [XAXYC21] (Paillier) | 1 | 3 | 6.3† | 226† | ✓ |
| [XALCCXYZ23] | 1 | 3 | 4.1† | 209† | ✓ |
| [DKLs24] | 2 | 3 | 115 | 29 | ✓ |
| [BHL24] | 1‡ | 2 | 5.6‡ | 144‡ | ✓ |
| **This Work** | 1 | 2 | 2 | 48 | ✓ |

Benchmarks were run on an Intel(R) Core(TM) i7-1365U CPU, 1 thread

# Disclaimer


The talk will **not focus** on inner technical details

The talk **will focus** on the protocol and optimizations

# Agenda



Fireblocks

# Agenda



Fireblocks

# ECDSA

**Setup:** $(\mathbb{G}, g, q)$
**Key Generation:**

- *Private Key:* $x \leftarrow \mathbb{Z}_q$
- *Public key:* $X = g^x \in \mathbb{G}$

**Sign:**

- *Input:* message $m$, Private Key: $x$
- $k \in \mathbb{Z}_q, \quad R = g^k, \quad r = R|_x$
- $\sigma = (\mathsf{H}(m) + x \cdot r) \cdot k^{-1} \bmod q$
- $(r, \sigma)$

**Verify:**

- message $m$, signature $(r, \sigma)$
- $\left( X^{\frac{r}{s}} \cdot g^{\frac{\mathsf{H}(m)}{s}} \right)\Big|_x \stackrel{?}{=} r$

# Oblivious Linear Evaluation (a.k.a. OLE)


$$y = a \cdot x + b$$

No one learns anything about the secrets of the other!

# Paillier Encryption

**KeyGen:**

- $(p, q)$ prime numbers, $N = pq$
- Public: $(N)$ Private: $(p, q, \varphi(N))$

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output

$$C = \mathrm{enc}_N(m; \rho) = (1 + mN) \cdot \rho^N \mod N^2.$$

**Decrypt:**

- On input $C \in \mathbb{Z}_{N^2}^*$ and $\varphi(N)$, output $m = \left(\frac{[C^{\varphi(N)}]_{N^2} - 1}{N}\right) \cdot \varphi(N)^{-1} \mod N$

: Computationally intensive

Additional ZK proofs (adapted from [CGGMP20]):

- $\Pi_{fac}$: N has no factor larger than $c \cdot \sqrt{N}$ with *c* "small" ($\sim 2^{l+l/2}$)
- $\Pi_{mod}$: N is the product of 2 primes congruent to *3 mod 4*

[CGGMP20] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. "UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts". In: ACM CCS 2020

# Instantiation of OLE with Paillier

- Comes from partial homomorphic property of Paillier's encryption:
  - Additively homomorphic
    - $\mathsf{Enc}_N(m_1) \times \mathsf{Enc}_N(m_2) \overset{\sim}{=} \mathsf{Enc}_N(m_1 + m_2)$
  - External product
    - $\mathsf{Enc}_N(m_1 \times m_2) \overset{\sim}{=} \mathsf{Enc}_N(m_1)^{m2}$

- Requires ZK Proofs $\Pi_{\text{fac}}$ and $\Pi_{\text{mod}}$

# Damgård-Fujisaki (2-)Commitments

**Setup:**

- $(p, q)$ strong prime numbers, $N = pq$
- $v \in \mathtt{QR}(N)$, $\lambda_1, \lambda_2 \in \mathbb{Z}_{\varphi(N)}$
- $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$
- Public: $(N, u_1, u_2)$ Private: $(p, q, \lambda_1, \lambda_2)$

**Parameters well-formedness proof:**

*Secret Input.* P holds $\lambda_1, \lambda_2 \in [\varphi(N)]$ such that $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$.

1. P sends $A \leftarrow v^{\alpha} \mod N$ for $\alpha \xleftarrow{\$} [\varphi(N)]$.
2. V replies with random challenges $e_1 \xleftarrow{\$} \{0,1\}$, $e_2 \xleftarrow{\$} \{0,1\}$.
3. P returns $z \leftarrow \alpha + e_1\lambda_1 + e_2\lambda_2 \mod \varphi(N)$.

*Verification.* V accepts if $v^z = A \cdot u_1^{e_1} \cdot u_2^{e_2} \mod N$

**Commitment:**

For $m_1, m_2 \in \mathbb{Z}$ and $(N, v, u_1, u_2)$, sample $\rho \leftarrow [N \cdot 2^{\ell}]$ and output

$$C = u_2^{m_1} \cdot u_2^{m_2} \cdot v^{\rho} \mod N.$$

: Computationally intensive

: Bandwidth consuming

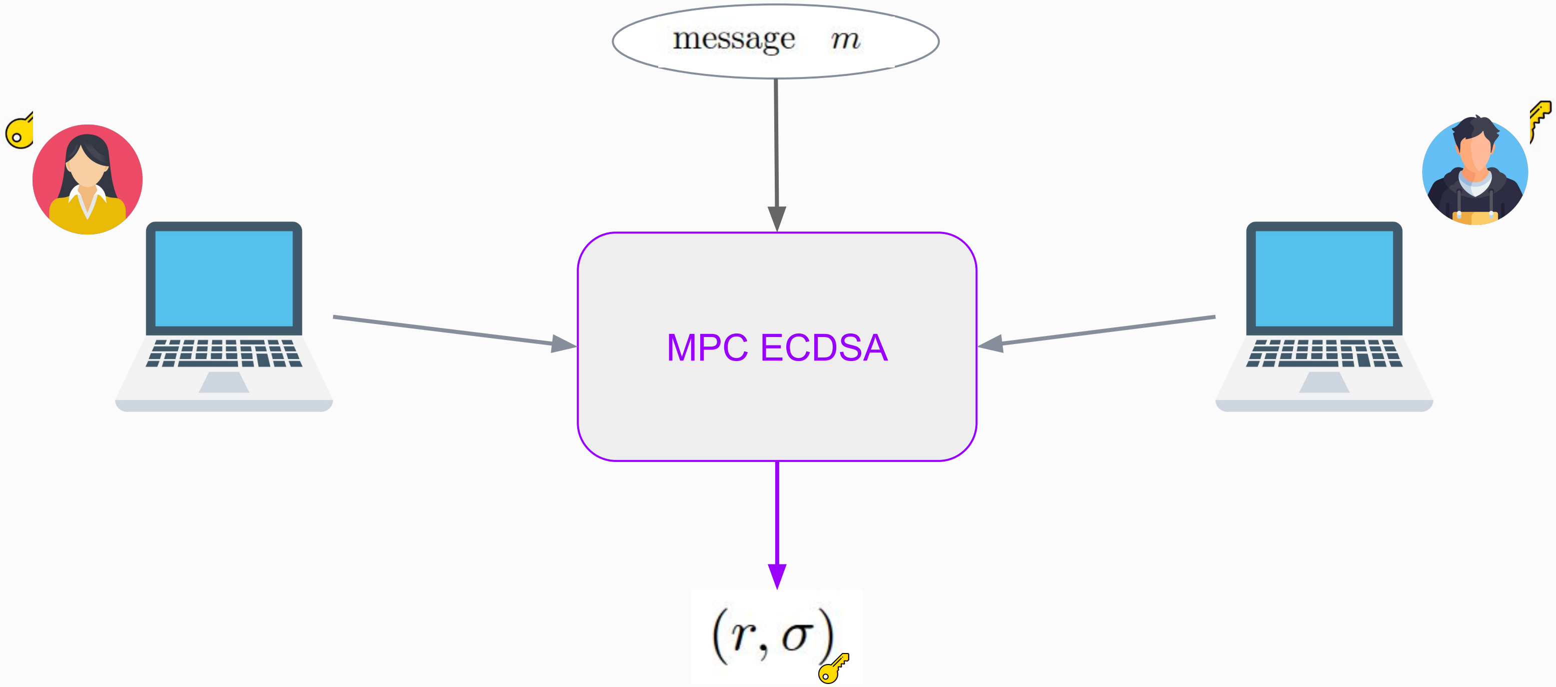
Fireblocks

# Agenda

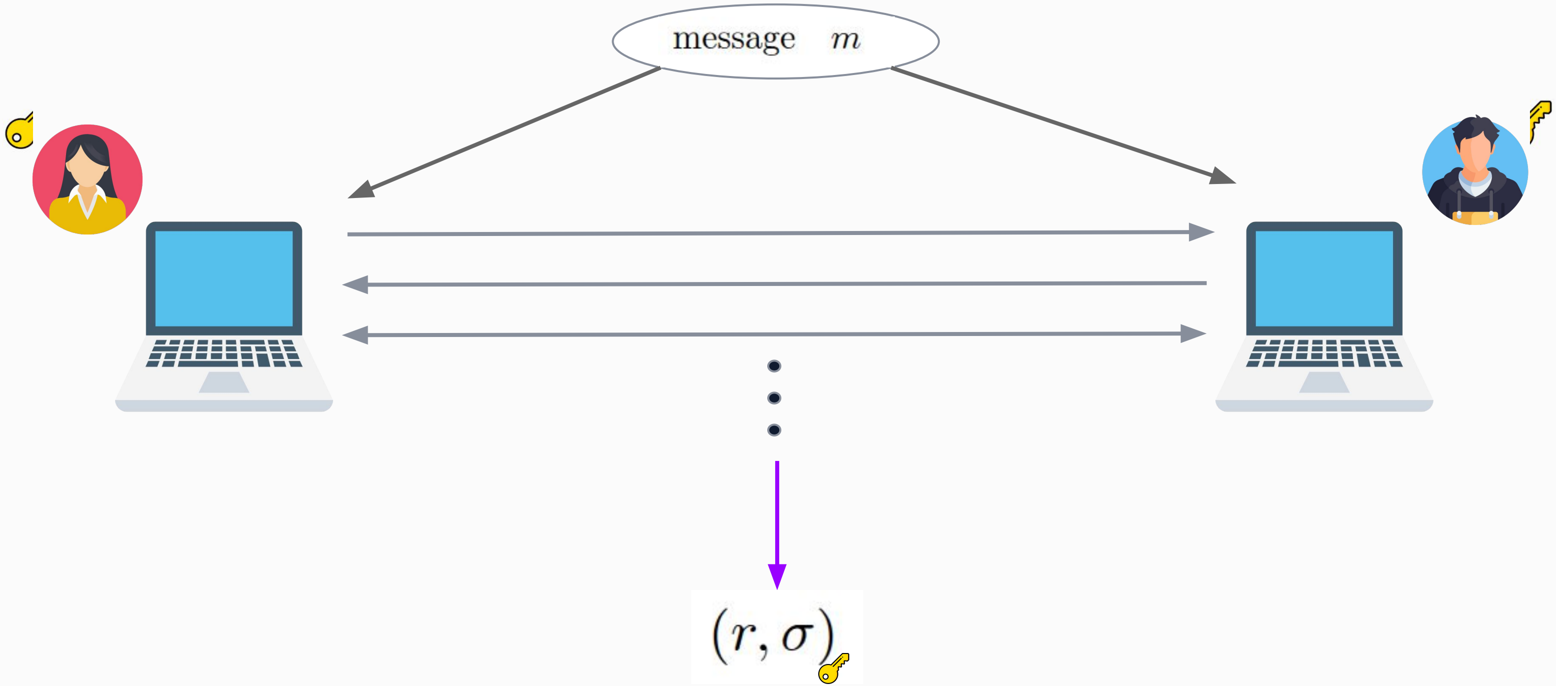


Fireblocks

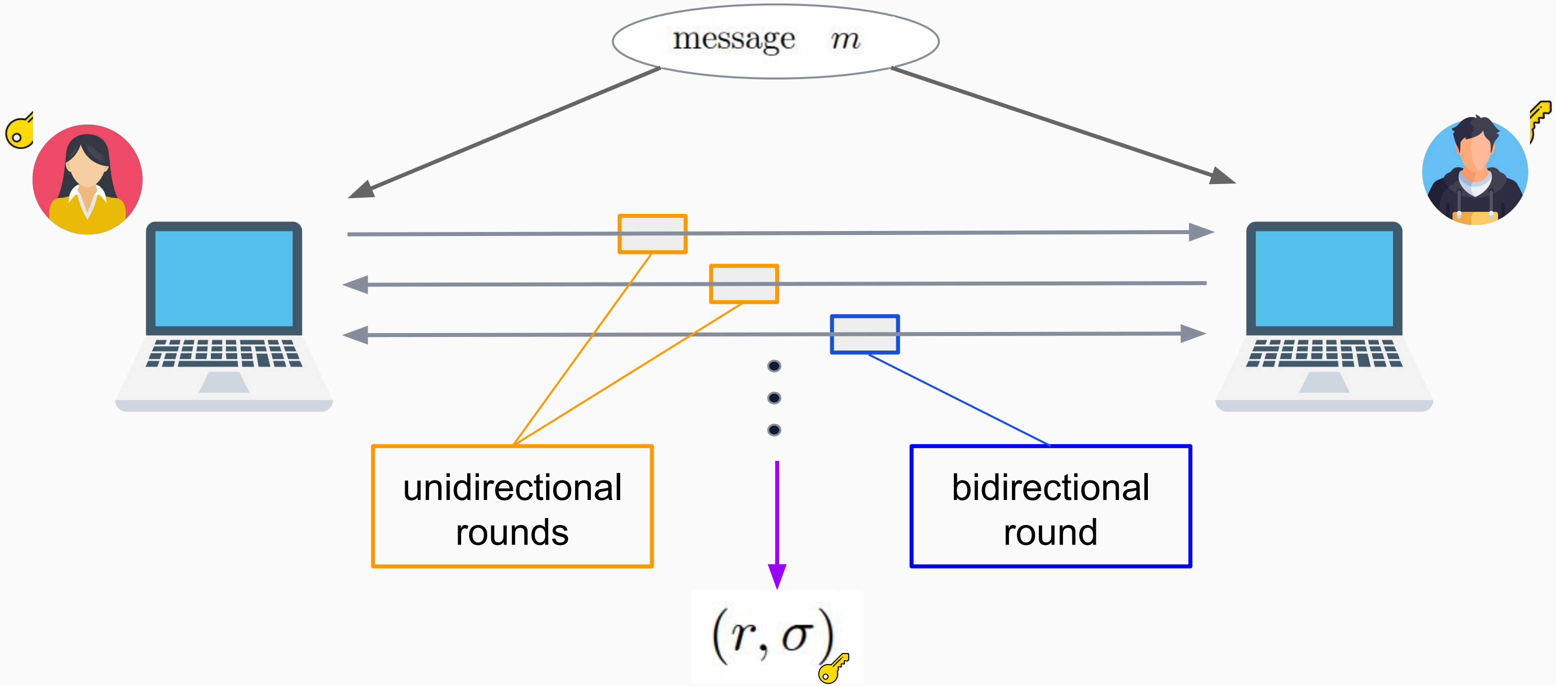
# 2-Party ECDSA



# 2-Party ECDSA



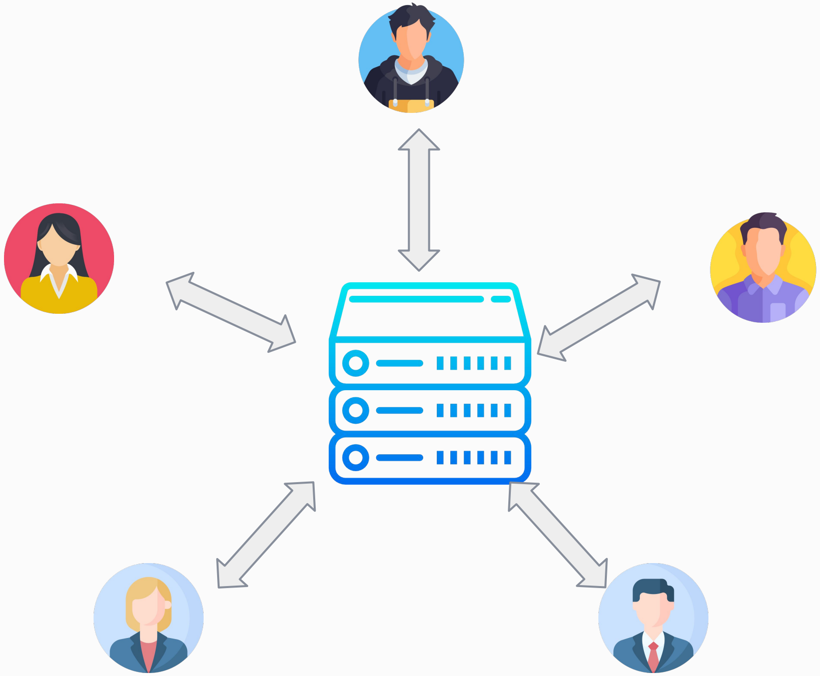
# 2-Party ECDSA



# Our 2-Party ECDSA

Preferred Setting:
Star-shaped topology

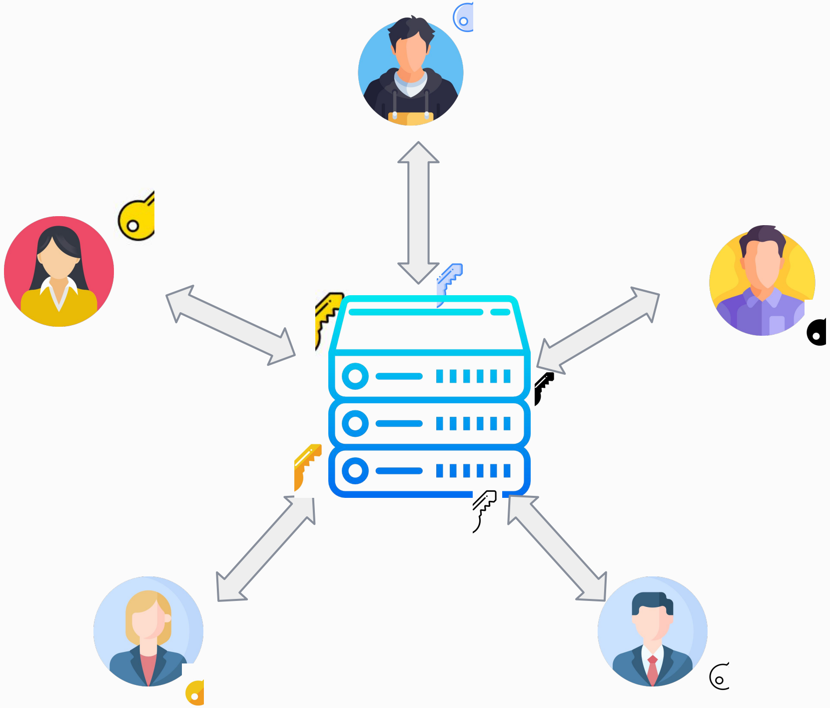
E.g. cryptocurrency wallet provider



# Our 2-Party ECDSA

Preferred Setting:
Star-shaped topology

E.g. cryptocurrency wallet provider



# Step 1 - Offline setup

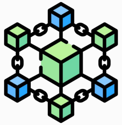
Elliptic Curve: $(\mathbb{G}, g)$

Generates:

- Paillier Key $N = pq$
  - ZK Proofs $\Pi_{\text{fac}}$ and $\Pi_{\text{mod}}$
- Damgård-Fujisaki setup $\left(\hat{N}, t, s_1, s_2\right)$
  - ZK Proof “Parameters Well-formedness” $\Pi_{\text{df}}$
- *(f, h)* two *nothing-up-my-sleeve* points on $\mathbb{G}$
  - Can be achieved *e.g.* using a cryptographic hash function + [SW06]

[SW06] Shallue, A. and C. E. van de Woestijne, "Construction of Rational Points on Elliptic Curves over Finite Fields", In ANTS 2006

# Step 1-a : Offline setup

 / 

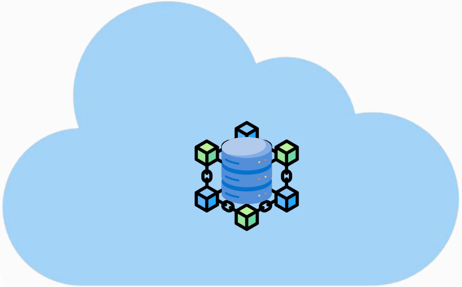
Elliptic Curve: $(\mathbb{G}, g)$

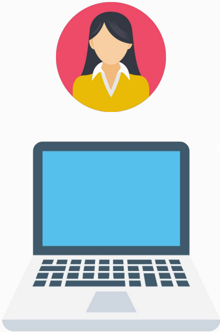
Generates:

- Paillier Key $N = pq$
  - ZK Proofs $\Pi_{\text{fac}}$ and $\Pi_{\text{mod}}$
- Damgård-Fujisaki setup $(\hat{N}, t, s_1, s_2)$
  - ZK Proof “Parameters Well-formedness” $\Pi_{\text{df}}$
- *(f, h)* two *nothing-up-my-sleeve* points on $\mathbb{G}$
  - Can be achieved *e.g.* using a cryptographic hash function + [SW06]

[SW06] Shallue, A. and C. E. van de Woestijne, "Construction of Rational Points on Elliptic Curves over Finite Fields", In ANTS 2006

# Step 1-b : Client registration





$\left(\mathbb{G}, g, N, \Pi_{\mathsf{mod}}, \Pi_{\mathsf{fac}}, \hat{N}, t, s_1, s_2, \Pi_{\mathsf{df}}, f, h\right)$

$$\mathbf{Verify}(\Pi_{\mathsf{mod}}) \stackrel{?}{=} 1$$

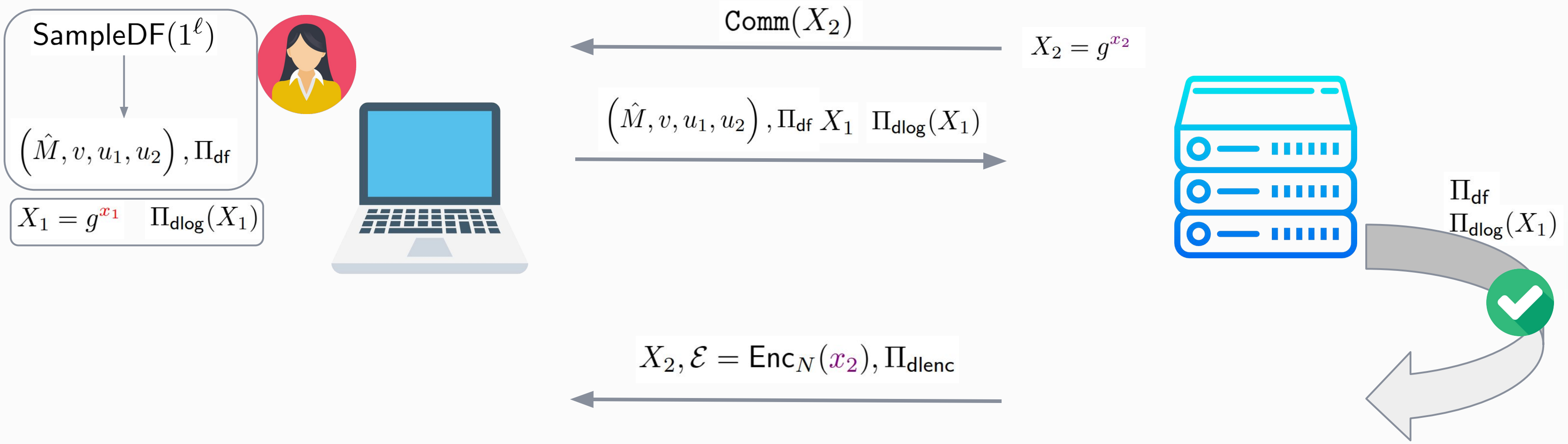
$$\mathbf{Verify}(\Pi_{\mathsf{fac}}) \stackrel{?}{=} 1$$

$$\mathbf{Verify}(\Pi_{\mathsf{df}}) \stackrel{?}{=} 1$$

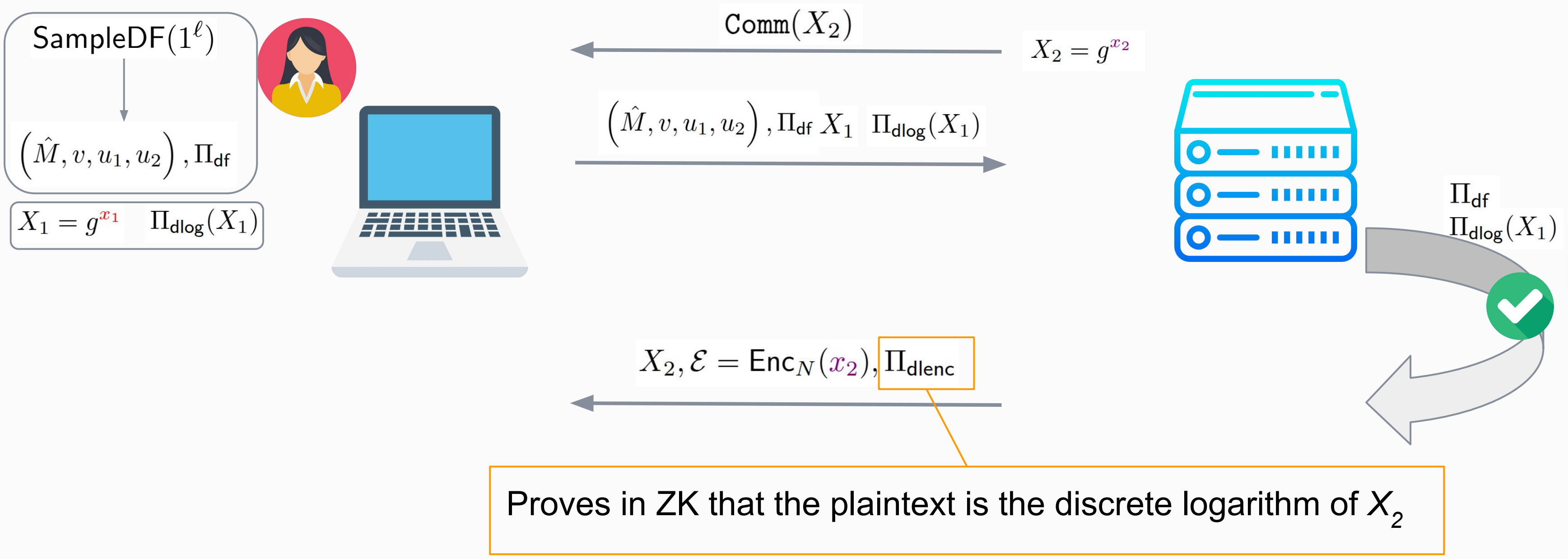
*Regenerate (f, h)* on $\mathbb{G}$

Stores $\left(\mathbb{G}, g, N, \hat{N}, t, s_1, s_2, f, h\right)$

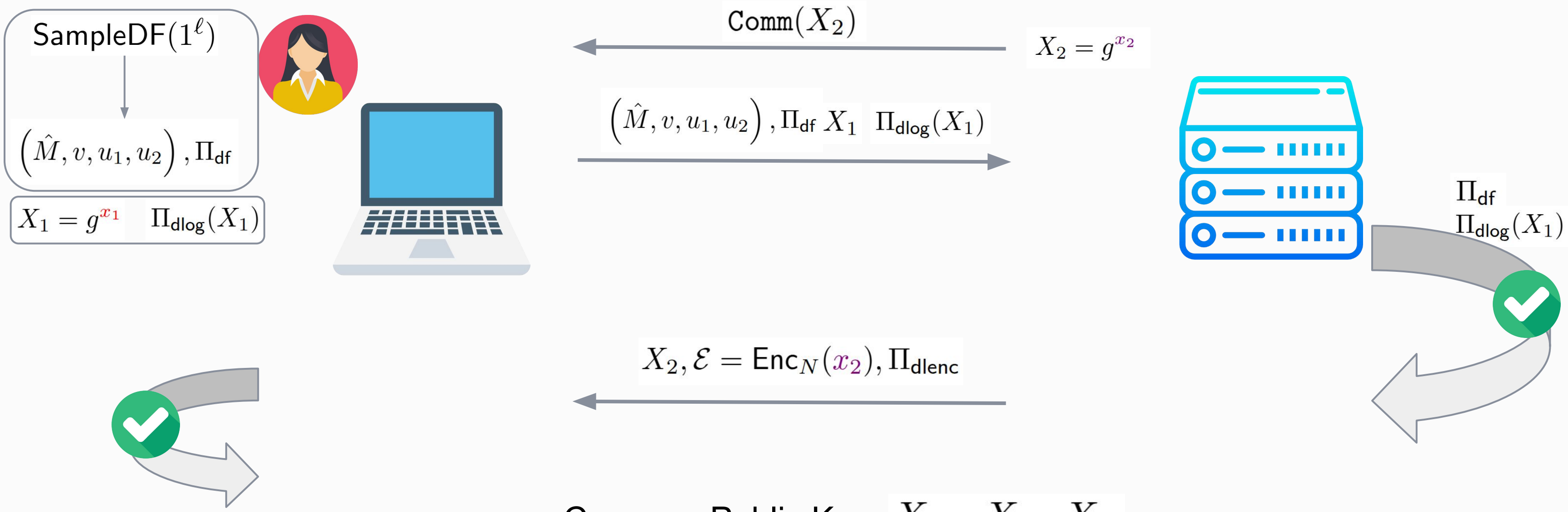
# Step 2: Key Generation



# Step 2: Key Generation



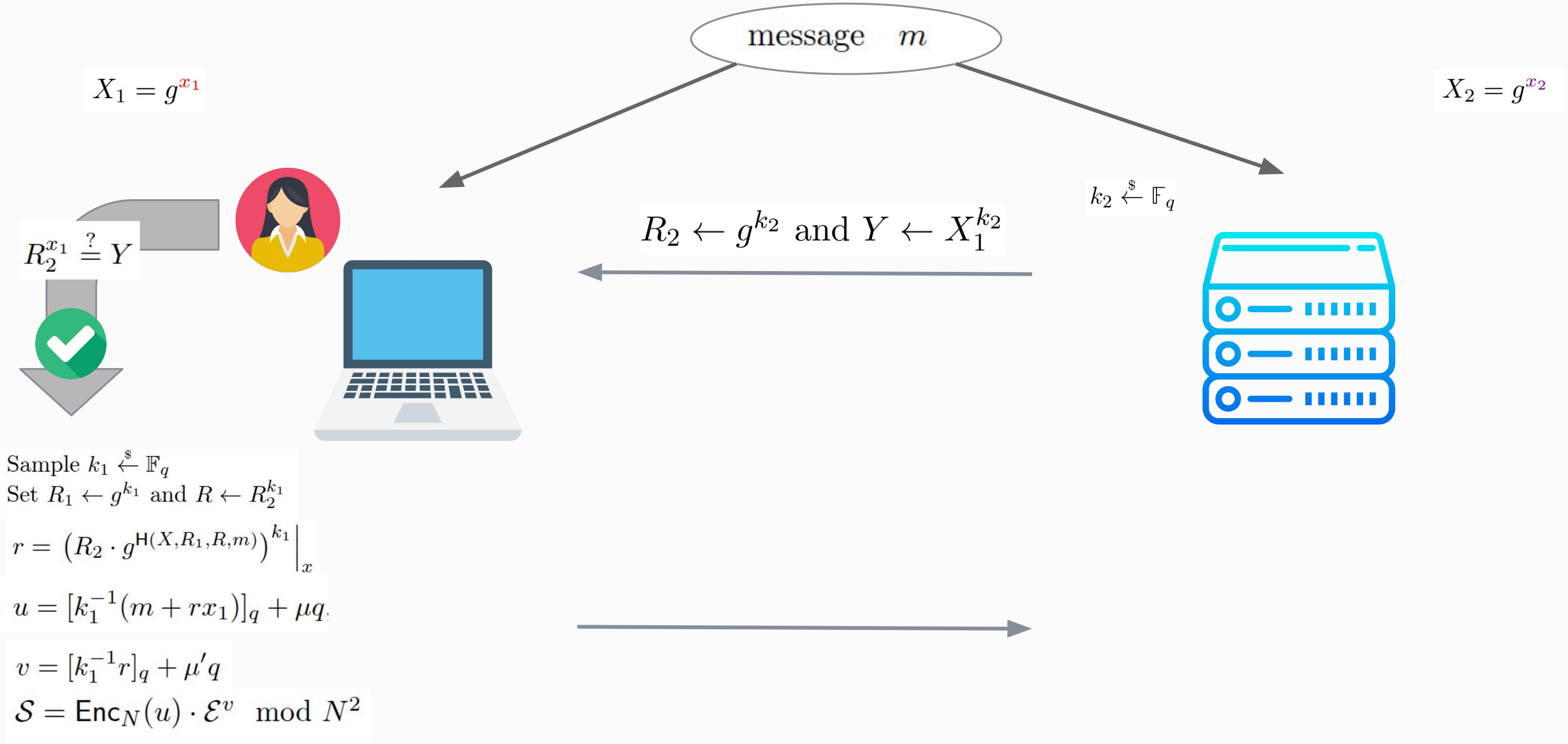
# Step 2: Key Generation



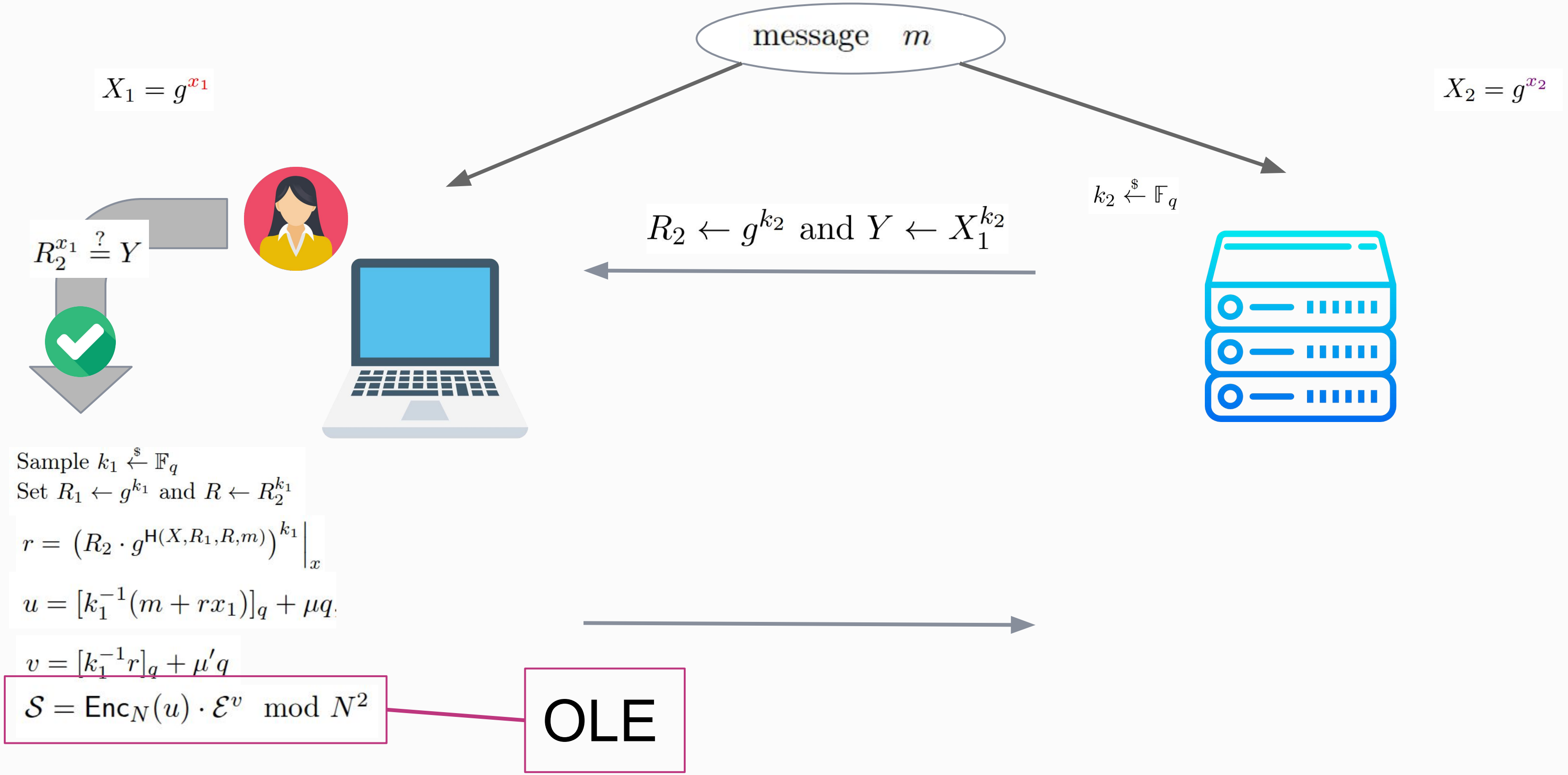

Common Public Key: $X = X_1 \cdot X_2$
Common Private Key: $x = x_1 + x_2$

Fireblocks

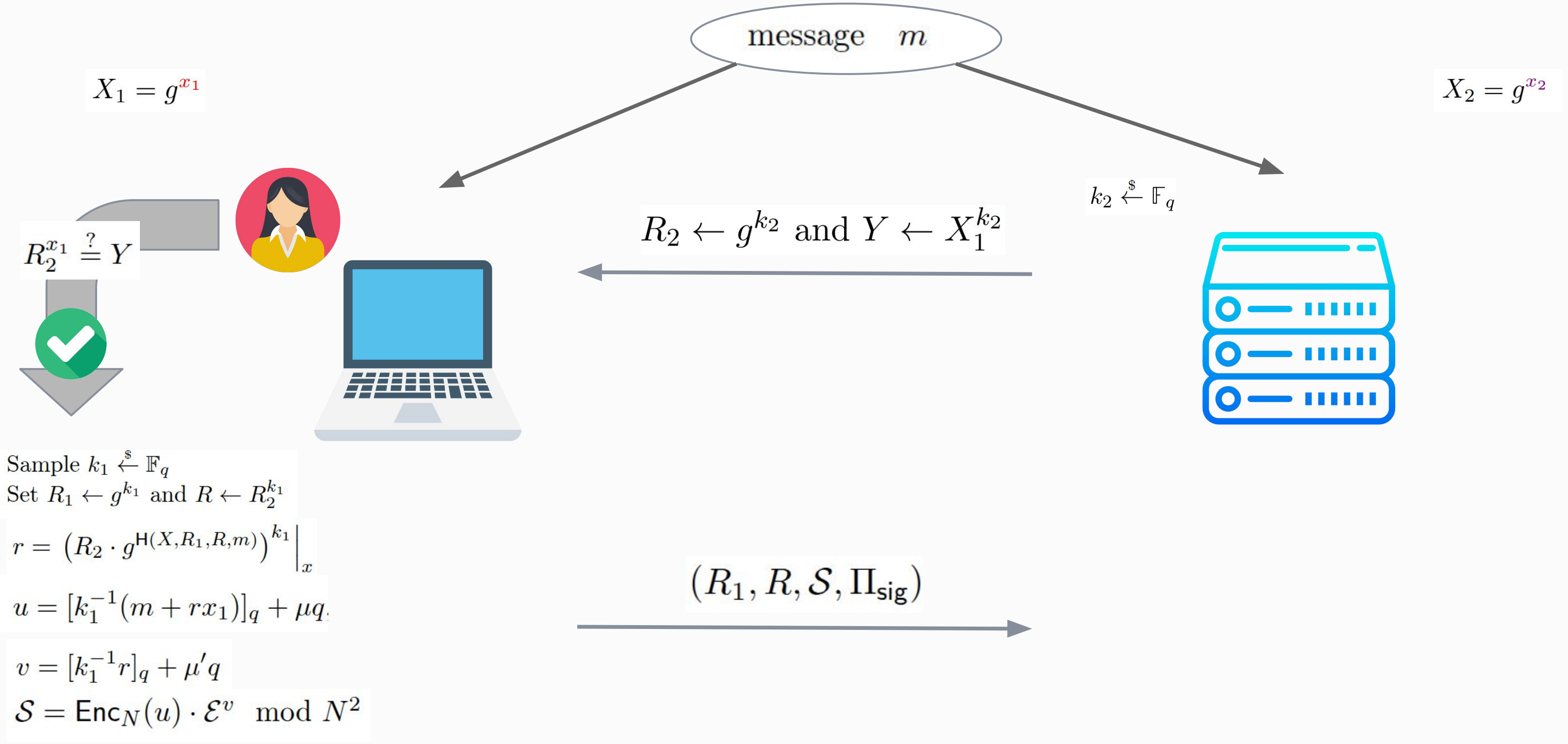
# Step 3: Signature Generation



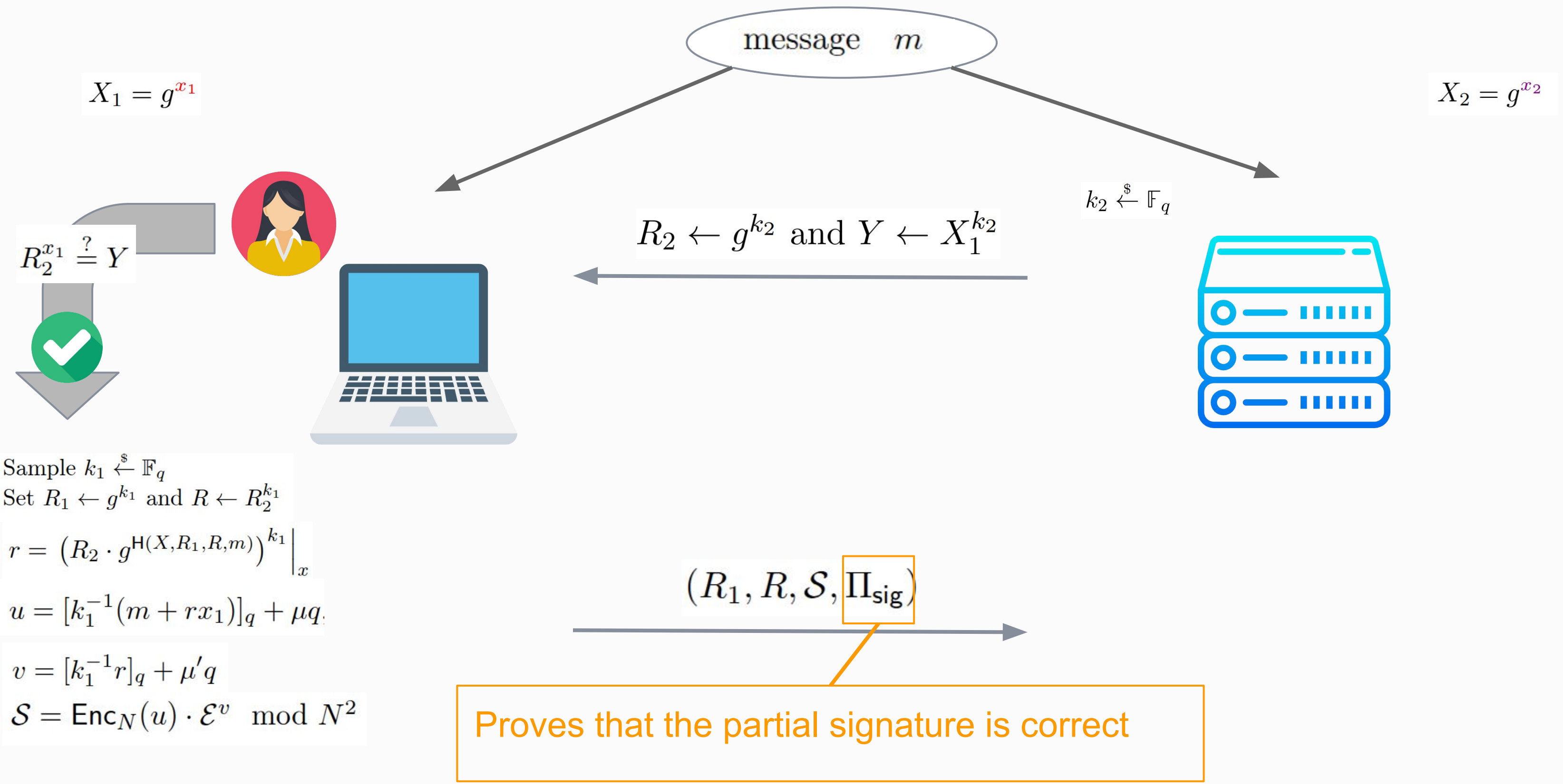
# Step 3: Signature Generation



# Step 3: Signature Generation



# Step 3: Signature Generation




Sample $k_1 \xleftarrow{\$} \mathbb{F}_q$

Set $R_1 \leftarrow g^{k_1}$ and $R \leftarrow R_2^{k_1}$

$$r = \left(R_2 \cdot g^{\mathsf{H}(X, R_1, R, m)}\right)^{k_1}\Big|_x$$

$$u = [k_1^{-1}(m + rx_1)]_q + \mu q$$

$$v = [k_1^{-1} r]_q + \mu' q$$

$$\mathcal{S} = \mathsf{Enc}_N(u) \cdot \mathcal{E}^v \mod N^2$$

# Step 3: Signature Generation

message $m$

$X_1 = g^{x_1}$

$X_2 = g^{x_2}$

$k_2 \xleftarrow{\$} \mathbb{F}_q$

$R_2 \leftarrow g^{k_2}$ and $Y \leftarrow X_1^{k_2}$

$R_2^{x_1} \stackrel{?}{=} Y$

Sample $k_1 \xleftarrow{\$} \mathbb{F}_q$

Set $R_1 \leftarrow g^{k_1}$ and $R \leftarrow R_2^{k_1}$

$r = \left(R_2 \cdot g^{\mathsf{H}(X, R_1, R, m)}\right)^{k_1}\Big|_x$

$u = [k_1^{-1}(m + r x_1)]_q + \mu q$

$v = [k_1^{-1} r]_q + \mu' q$

$\mathcal{S} = \mathsf{Enc}_N(u) \cdot \mathcal{E}^v \mod N^2$

$(R_1, R, \mathcal{S}, \Pi_{\mathsf{sig}})$

$R_1 \neq \mathbb{1}$, $\mathcal{S} \in \mathbb{Z}_{N^2}^*$ and $R = R_1^{k_2}$

$\mathbf{Verify}(\Pi_{\mathsf{sig}}) \stackrel{?}{=} 1$

Publish the following signature

$r, \sigma = \mathsf{Dec}_{\varphi(N)}(\mathcal{S}) \cdot (k_2 + (X, R_1, R, m))^{-1} \mod q$

iff it verifies

# Agenda



Fireblocks

# A new class of prime numbers: Tough Primes

- Strong prime
  - $p = 2.p_0 + 1$ with $p_0$ prime

- Tough prime
  - $l$ security parameter
  - $p = 2.p_0.p_1 \ldots p_t + 1$ where $p_i$ $2l$-bits primes
  - Ex: if $l$=128, $p_i$ are 256-bits primes

- To the best of our knowledge, it's the first time this is proposed

Claim: Tough primes provide **identical security guarantees**, compared to strong primes, in a variety of contexts e.g. Damgard-Fujisaki commitment, ZK proofs, …

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

| Bit size \ Prime type | Regular | | Tough | | Strong | |
|---|---|---|---|---|---|---|
| | average | std dev. | average | std dev. | average | std dev. |
| 1024 | 23.2 | 12.6 | 37.1 | 18.4 | 586.7 | 555.8 |
| 1536 | 70.5 | 35.9 | 98.9 | 53.9 | $3.6\times10^3$ | $4.2\times10^3$ |
| 2048 | 204.7 | 137.1 | 235.5 | 143.1 | $14.8\times10^3$ | $15.3\times10^3$ |

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

| Bit size \ Prime type | Regular | | Tough | | Strong | |
|---|---|---|---|---|---|---|
| | average | std dev. | average | std dev. | average | std dev. |
| 1024 | 23.2 | 12.6 | 37.1 | 18.4 | 586.7 | 555.8 |
| 1536 | $\boxed{70.5}$ | 35.9 | $\boxed{98.9}$ | 53.9 | $3.6\times10^3$ | $4.2\times10^3$ |
| 2048 | 204.7 | 137.1 | 235.5 | 143.1 | $14.8\times10^3$ | $15.3\times10^3$ |

x1.4

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

| Prime type / Bit size | Regular | | Tough | | Strong | |
|---|---|---|---|---|---|---|
| | average | std dev. | average | std dev. | average | std dev. |
| 1024 | 23.2 | 12.6 | 37.1 | 18.4 | 586.7 | 555.8 |
| 1536 | $\boxed{70.5}$ | 35.9 | 98.9 | 53.9 | $\boxed{3.6\times10^3}$ | $4.2\times10^3$ |
| 2048 | 204.7 | 137.1 | 235.5 | 143.1 | $14.8\times10^3$ | $15.3\times10^3$ |

x50

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

| Bit size \ Prime type | Regular | | Tough | | Strong | |
|---|---|---|---|---|---|---|
| | average | std dev. | average | std dev. | average | std dev. |
| 1024 | 23.2 | 12.6 | 37.1 | 18.4 | 586.7 | 555.8 |
| 1536 | 70.5 | 35.9 | 98.9 | 53.9 | $3.6\times10^3$ | $4.2\times10^3$ |
| 2048 | 204.7 | 137.1 | 235.5 | 143.1 | $14.8\times10^3$ | $15.3\times10^3$ |

x1.2

# Generation of Tough Primes

*Input.* Security parameter $\ell$ and bit-length $n$ s.t. $2\ell$ divides $n$.
Let $t \leftarrow n/(2\ell)$.

*Operation.*

1. Sample a pool $\mathbf{B}$ of $2^{2\ell}$-sized primes.
2. Enumerate over all unordered combinations of $t$ primes in $\mathbf{B}$.
   - Let $\{p_1, \ldots, p_t\}$ be the combination at any given iteration.
   - If $P \leftarrow 2 \cdot p_1 \cdots p_t + 1$ is a prime number, break the loop.

*Output.* $P$

| Prime type / Bit size | Regular | | Tough | | Strong | |
|---|---|---|---|---|---|---|
| | average | std dev. | average | std dev. | average | std dev. |
| 1024 | 23.2 | 12.6 | 37.1 | 18.4 | 586.7 | 555.8 |
| 1536 | 70.5 | 35.9 | 98.9 | 53.9 | $3.6\times10^3$ | $4.2\times10^3$ |
| 2048 | $\boxed{204.7}$ | 137.1 | 235.5 | 143.1 | $\boxed{14.8\times10^3}$ | $15.3\times10^3$ |

x72

Fireblocks

# Use of Tough Primes in our Protocol

- Generation of Damgård-Fujisaki parameters
  - Server's side during the setup
  - Client's side during key generation

- 14x improvement in Setup generation

- 13x improvement in key generation time for the client

# Short Exponents (SE)

- SEDL: SE Discrete Logarithm

**Definition 3.4** (SEDL over $\mathbb{Z}_N^*$). Define SEDL such that $(N, t, [t^x]_N) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)$ for $(N; p, q) \xleftarrow{\$} \mathsf{SampleRSA}(1^\ell)$, $t \xleftarrow{\$} \mathsf{QR}_N$ and $x \xleftarrow{\$} \pm 2^{2\ell}$. We say that *small-exponent discrete-log (SEDL) holds true* if for every PPTM A, there exists a negligible function $\mu : \mathbb{N} \to \mathbb{R}$ such that

$$\Pr_{(N,t,s) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)} \left[ x_0 \xleftarrow{\$} \mathsf{A}(1^\ell, N, t, s) \text{ s.t. } t^{x_0} = s \bmod N \right] \leq \mu(\ell).$$

- SEI: SE Indistinguishability

**Definition 3.5** (SEI over $\mathbb{Z}_N^*$). Define distribution ensemble SEI such that $(N, t, [t^{\alpha x + (1-\alpha)y}]_N, \alpha) \xleftarrow{\$} \mathsf{SEDL}(1^\ell)$ for $N \xleftarrow{\$} \mathsf{SampleRSA}(1^\ell)$, $t \xleftarrow{\$} \mathsf{QR}_N$, $x \xleftarrow{\$} \pm 2^{2\ell}$, $y \xleftarrow{\$} \pm N$ and $\alpha \xleftarrow{\$} \{0, 1\}$. We say that *small-exponent indistinguishability (SEI) holds true* if for every PPTM A, there exists a negligible function $\mu : \mathbb{N} \to \mathbb{R}$ such that

$$\Pr_{(N,t,s,\alpha) \xleftarrow{\$} \mathsf{SEI}(1^\ell)} \left[ \alpha_0 \xleftarrow{\$} \mathsf{A}(1^\ell, N, t, s) \text{ s.t. } \alpha_0 = \alpha \right] \leq \frac{1}{2} + \mu(\ell).$$

- Shown that SEI reduces to SEDL in prime order group [KK04]. We showed it also reduces in the case of unknown order groups (e.g. RSA multiplicative group)

[KK04] Takeshi Koshiba and Kaoru Kurosawa. “Short Exponent Diffie-Hellman Problems”. In: PKC 2004.

# Use of Short Exponents: Paillier Encryption

- Paillier encryption

**Encrypt:**

- On input $m \in \mathbb{Z}_N$ and $N$, sample $\rho \leftarrow \mathbb{Z}_N^*$
- Output

$$C = \text{enc}_N(m; \rho) = (1 + mN) \cdot \rho^N \mod N^2.$$

  - Pick $\rho = \rho_0^{2N}$ => can be a constant to the Paillier parameters
    - Needs to be checked by the clients
  - $C = \text{enc}_N(m; \rho, r) = (1 + mN) \cdot \rho^r \mod N^2$ s.t. $r \xleftarrow{\$} [2^{2\ell}]$
    - Computation time linear to exponent bitlength

- Decryption is still possible since

$$(\rho^r)^{\varphi(N)} = (\rho_0^{2N})^{\varphi(N)} = \left(\rho_0^{N \cdot \varphi(N)}\right)^2 = 1^2 = 1$$

# Use of Short Exponents: Damgård-Fujisaki

**Setup:**

- $(p, q)$ strong prime numbers, $N = pq$
- $v \in \mathtt{QR}(N)$, $\lambda_1, \lambda_2 \in \mathbb{Z}_{\varphi(N)}$
- $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$
- Public: $(N, u_1, u_2)$ Private: $(p, q, \lambda_1, \lambda_2)$

**Parameters well-formedness proof:**

*Secret Input.* P holds $\lambda_1, \lambda_2 \in [\varphi(N)]$ such that $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$.

1. P sends $A \leftarrow v^{\alpha} \mod N$ for $\alpha \xleftarrow{\$} [\varphi(N)]$.
2. V replies with random challenges $e_1 \xleftarrow{\$} \{0,1\}, e_2 \xleftarrow{\$} \{0,1\}$.
3. P returns $z \leftarrow \alpha + e_1\lambda_1 + e_2\lambda_2 \mod \varphi(N)$.

*Verification.* V accepts if $v^z = A \cdot u_1^{e_1} \cdot u_2^{e_2} \mod N$

**Commitment:**

For $m_1, m_2 \in \mathbb{Z}$ and $(N, v, u_1, u_2)$, sample $\rho \leftarrow [N \cdot 2^{\ell}]$ and output

$$C = u_2{}^{m_1} \cdot u_2{}^{m_2} \cdot v^{\rho} \mod N.$$

: Computationally intensive

: Bandwidth consuming

# Use of Short Exponents: Damgård-Fujisaki

**Setup:**

- $(p, q)$ strong prime numbers, $N = pq$
- $v \in \mathtt{QR}(N)$, $\lambda_1, \lambda_2 \in \mathbb{Z}_{\varphi(N)}$
- $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$
- Public: $(N, u_1, u_2)$ Private: $(p, q, \lambda_1, \lambda_2)$

**Parameters well-formedness proof:**

*Secret Input.* P holds $\lambda_1, \lambda_2 \in [\varphi(N)]$ such that $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$.

1. P sends $A \leftarrow v^{\alpha} \mod N$ for $\alpha \xleftarrow{\$} [\varphi(N)]$.
2. V replies with random challenges $e_1 \xleftarrow{\$} \{0,1\}$, $e_2 \xleftarrow{\$} \{0,1\}$.
3. P returns $z \leftarrow \alpha + e_1\lambda_1 + e_2\lambda_2 \mod \varphi(N)$.

*Verification.* V accepts if $v^z = A \cdot u_1^{e_1} \cdot u_2^{e_2} \mod N$

Can be accelerated thanks to short exponent trick

$N, v, u_1, u_2)$, sample $\rho \leftarrow [N \cdot 2^{\ell}]$ and output

$$C = u_2^{m_1} \cdot u_2^{m_2} \cdot v^{\rho} \mod N.$$

: Computationally intensive

: Bandwidth consuming

# Use of Short Exponents: Damgård-Fujisaki

**Setup:**

- $(p, q)$ strong prime numbers, $N = pq$
- $v \in \mathtt{QR}(N)$, $\boxed{\lambda_1, \lambda_2 \in \mathbb{Z}_{\varphi(N)}}$
- $u_1 = \boxed{v^{\lambda_1}} \mod N$, $u_2 = \boxed{v^{\lambda_2}} \mod N$
- Public: $(N, u_1, u_2)$ Private: $(p, q, \lambda_1, \lambda_2)$

**Parameters well-formedness proof:**

*Secret Input.* P holds $\lambda_1, \lambda_2 \in [\varphi(N)]$ such that $u_1 = v^{\lambda_1} \mod N$, $u_2 = v^{\lambda_2} \mod N$.

1. P sends $\boxed{A \leftarrow v^{\alpha}} \mod N$ for $\boxed{\alpha \xleftarrow{\$} [\varphi(N)]}$.
2. V replies with random challenges $e_1 \xleftarrow{\$} \{0, 1\}$, $e_2 \xleftarrow{\$} \{0, 1\}$.
3. P returns $\boxed{z \leftarrow \alpha + e_1\lambda_1 + e_2\lambda_2 \mod \varphi(N)}$.

*Verification.* V accepts if $\boxed{v^z} = A \cdot u_1^{e_1} \cdot u_2^{e_2} \mod N$

Can be accelerated thanks to short exponent trick

$N, v, u_1, u_2)$, sample $\rho \leftarrow [N \cdot 2^{\ell}]$ and output

$$C = u_2^{m_1} \cdot u_2^{m_2} \cdot \boxed{v^{\rho}} \mod N.$$

: Computationally intensive : Bandwidth consuming

Fireblocks

# Use of Short Exponents: ZK Proofs

- 

# Agenda

01 Preliminaries

02 2PC ECDSA Protocol

03 Additional Techniques

04 **Open Questions & Future Work**

Fireblocks

# Work in Progress and Future Work

- [WIP] Protocol under deployment in real-world
- Stateless Server during signature
- Tough primes
  - Applications to other use cases