Secure Multiparty Computation from Threshold Encryption based on Class Groups

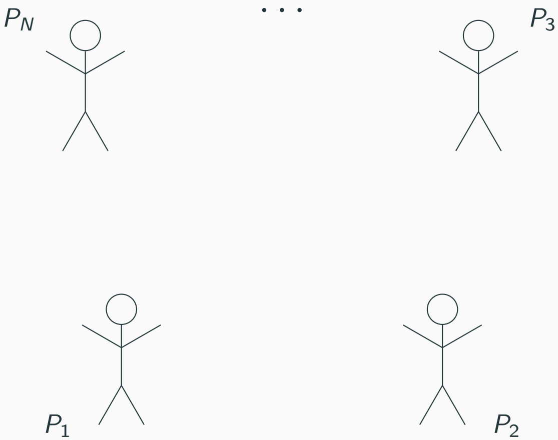
Lennart Braun, Ivan Damgård, and Claudio Orlandi

August 23, 2023 – Crypto'23

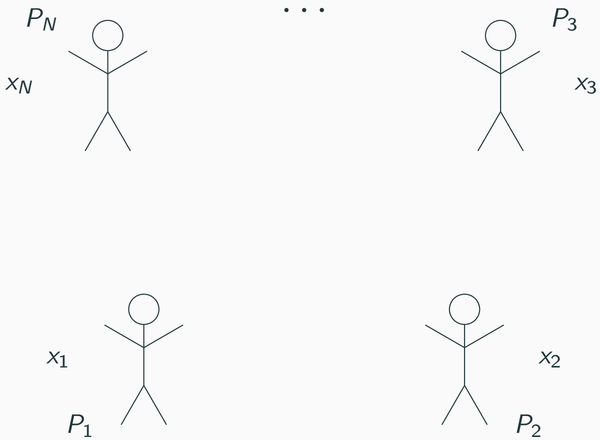
Aarhus University

Introduction and Preliminaries

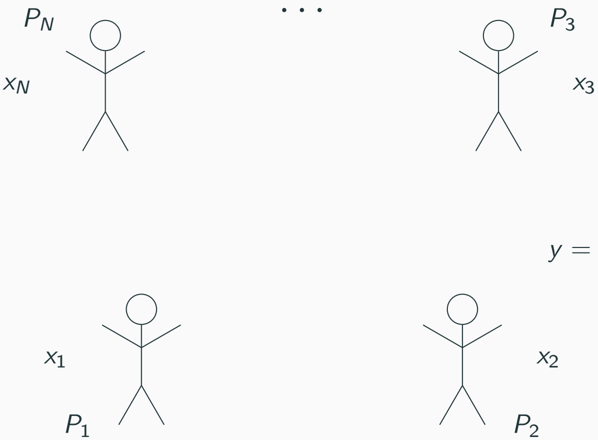
Secure Multiparty Computation



Secure Multiparty Computation

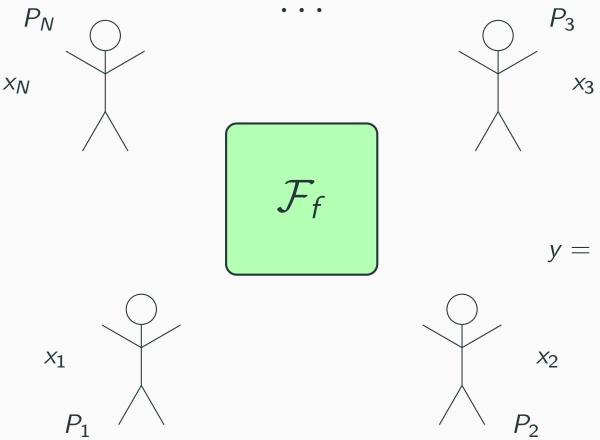


Secure Multiparty Computation



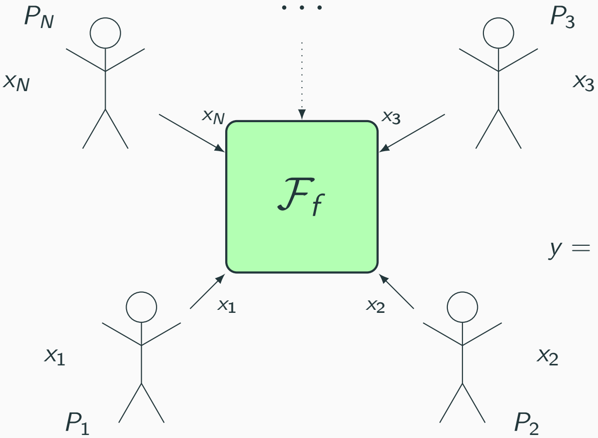
$$y = f(x_1, \ldots, x_N)$$

Secure Multiparty Computation



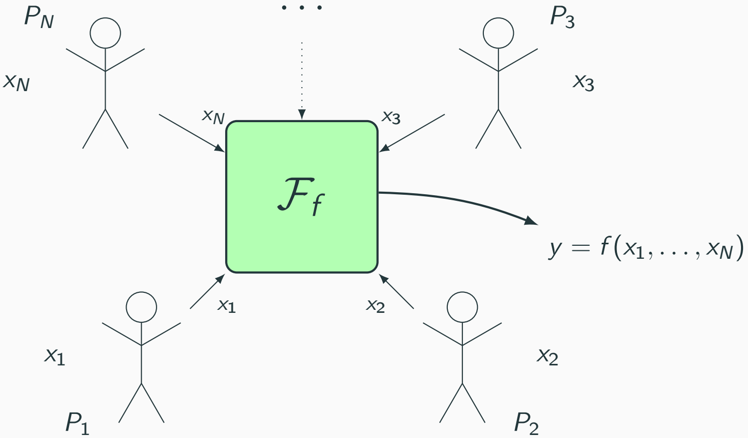
$$y = f(x_1, \ldots, x_N)$$

Secure Multiparty Computation

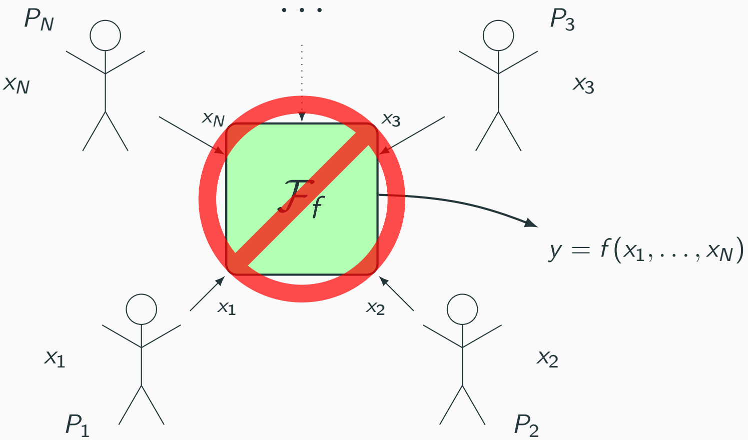


$$y = f(x_1, \ldots, x_N)$$

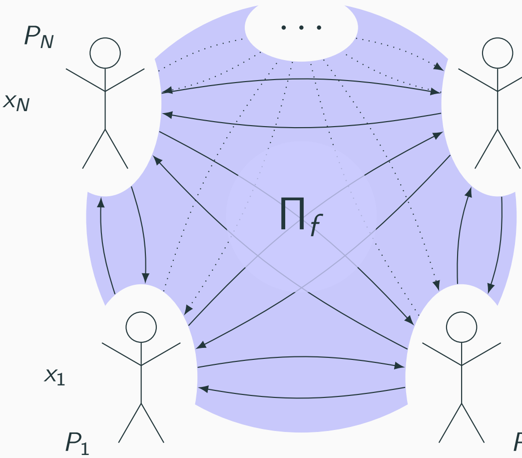
Secure Multiparty Computation



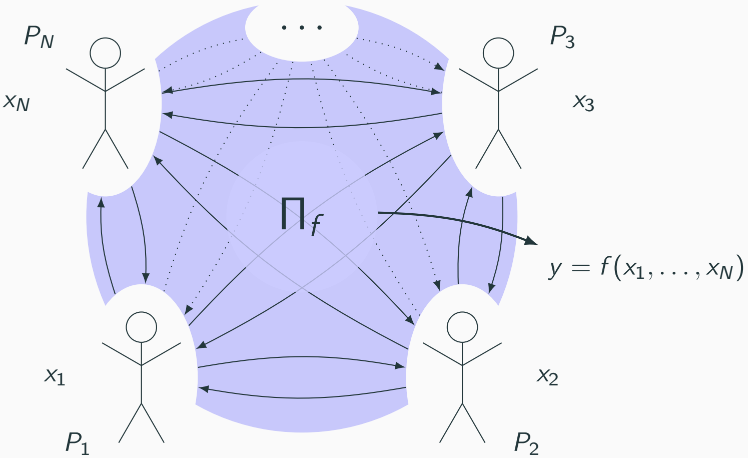
Secure Multiparty Computation



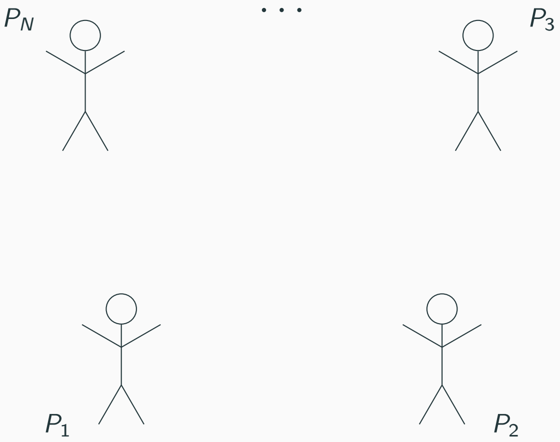
Secure Multiparty Computation



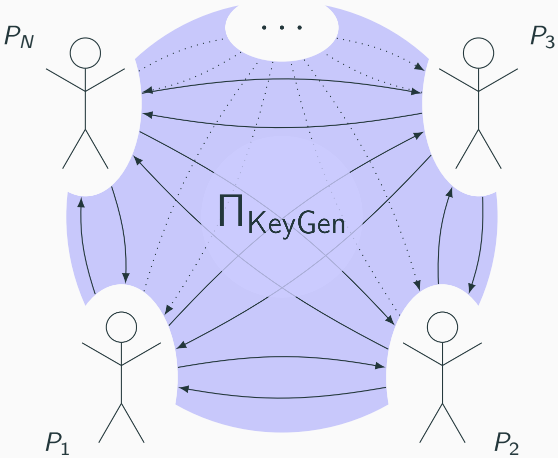
$y = f(x_1, \ldots, x_N)$


P_N
x_N
$\cdots$
P_3
x_3
Π_f
$y = f(x_1, \ldots, x_N)$
x_1
P_1
x_2
P_2

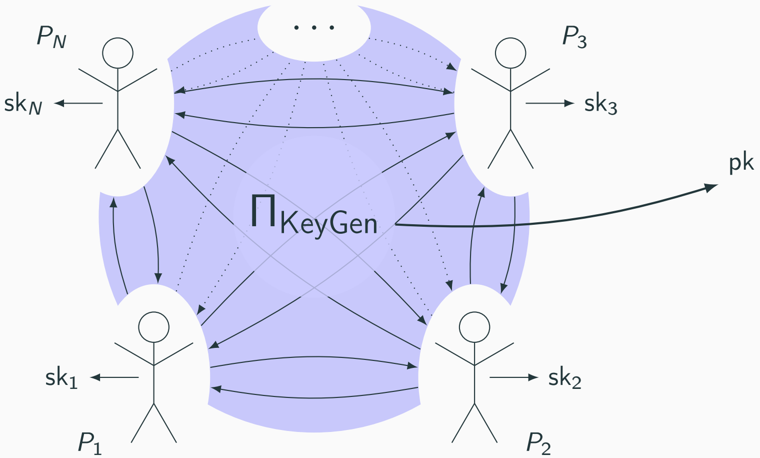
Threshold Encryption



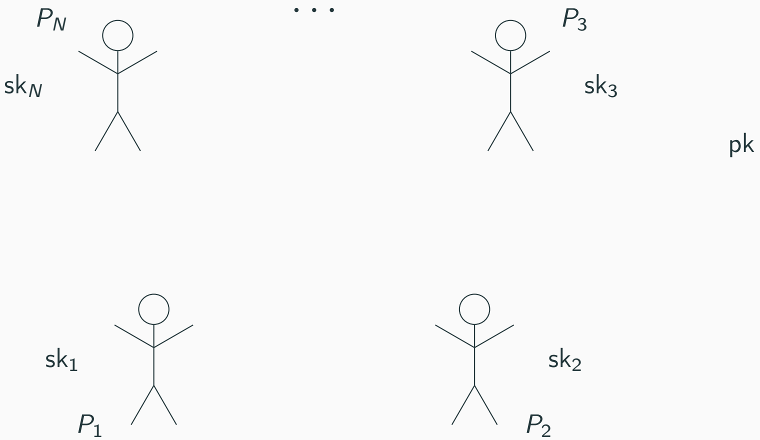
Threshold Encryption: Distributed Key Generation



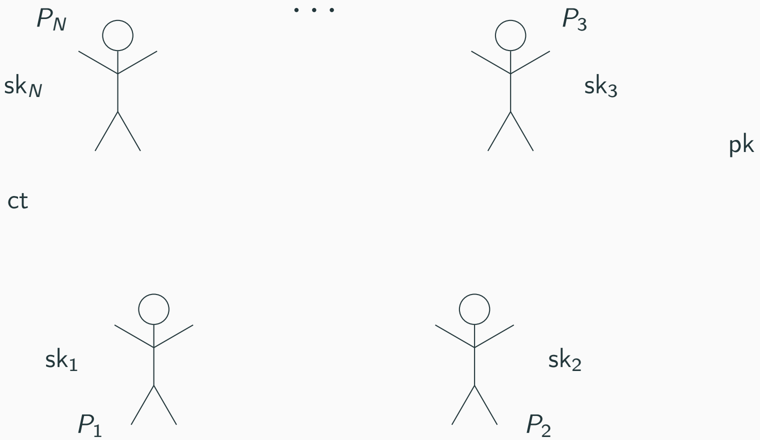
Threshold Encryption: Distributed Key Generation



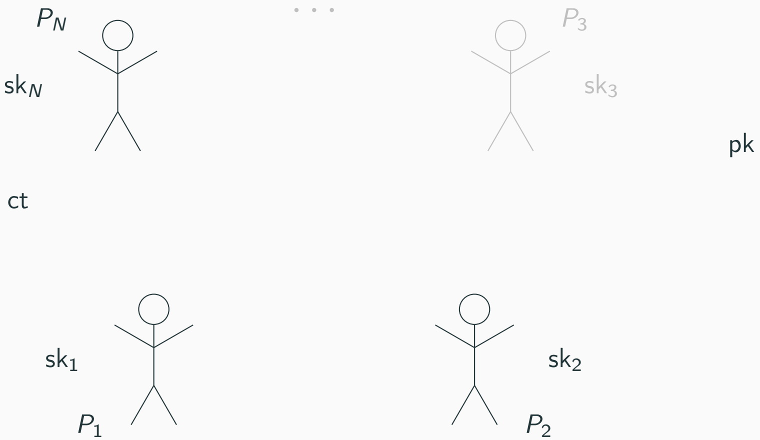
Threshold Encryption: Distributed Key Generation



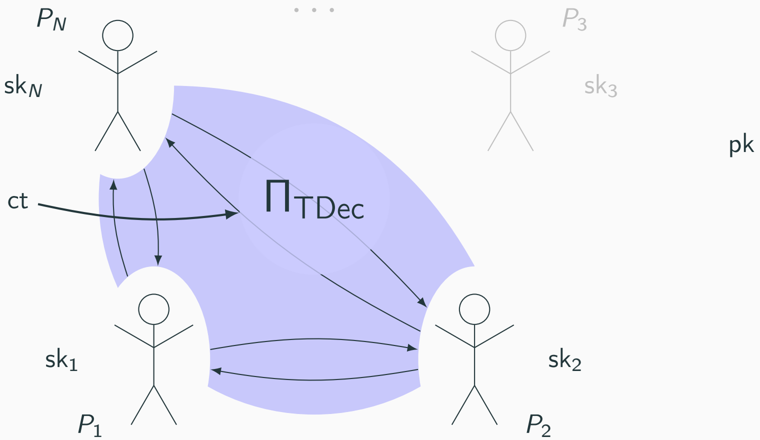
Threshold Encryption: Distributed Key Generation and Decryption



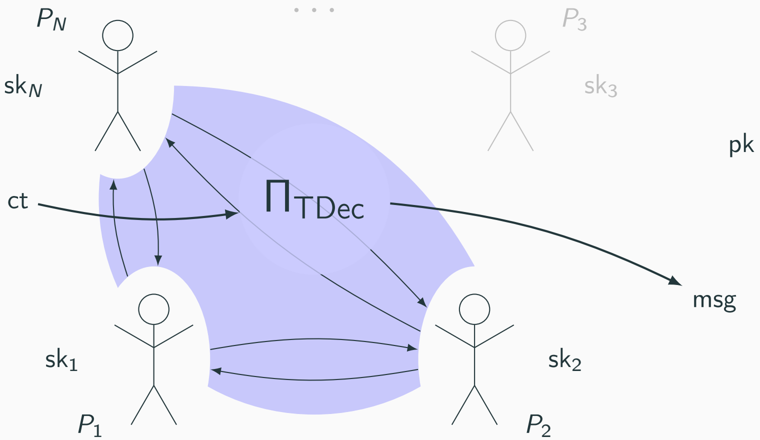
Threshold Encryption: Distributed Key Generation and Decryption



Threshold Encryption: Distributed Key Generation and Decryption



Threshold Encryption: Distributed Key Generation and Decryption



Class Groups

~~Class Groups~~ The CL Framework for Groups of Unknown Order [CL15]

Following Castagnos and Laguillaumie ([CL15] and follow-up works)

~~Class Groups~~ The CL Framework for Groups of Unknown Order [CL15]

Following Castagnos and Laguillaumie ([CL15] and follow-up works)

- $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$
  - 1^λ computational security parameter
  - $q > 2^\lambda$ prime

~~Class Groups~~ The CL Framework for Groups of Unknown Order [CL15]

Following Castagnos and Laguillaumie ([CL15] and follow-up works)

- $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$
  - 1^λ computational security parameter
  - $q > 2^\lambda$ prime
- Cyclic group $G \simeq G^q \times F$

~~Class Groups~~ The CL Framework for Groups of Unknown Order [CL15]

Following Castagnos and Laguillaumie ([CL15] and follow-up works)

- $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$
  - 1^λ computational security parameter
  - $q > 2^\lambda$ prime
- Cyclic group $G \simeq G^q \times F$
  - $F = \langle f \rangle$ – subgroup of order q with easy DLog

~~Class Groups~~ The CL Framework for Groups of Unknown Order [CL15]

Following Castagnos and Laguillaumie ([CL15] and follow-up works)

- $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$
  - 1^λ computational security parameter
  - $q > 2^\lambda$ prime
- Cyclic group $G \simeq G^q \times F$
  - $F = \langle f \rangle$ – subgroup of order q with easy DLog
  - $G^q = \langle g \rangle$ – subgroup of qth powers with unknown order

~~Class Groups~~ The CL Framework for Groups of Unknown Order [CL15]

Following Castagnos and Laguillaumie ([CL15] and follow-up works)

- $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$
  - 1^λ computational security parameter
  - $q > 2^\lambda$ prime
- Cyclic group $G \simeq G^q \times F$
  - $F = \langle f \rangle$ – subgroup of order q with easy DLog
  - $G^q = \langle g \rangle$ – subgroup of qth powers with unknown order
- Hardness assumptions
  - ORD: hard to find the order of any $h \in G \setminus F$
  - HSM: hard to distinguish random elements of G and G^q

~~Class Groups~~ The CL Framework for Groups of Unknown Order [CL15]

Following Castagnos and Laguillaumie ([CL15] and follow-up works)

- $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$
  - 1^λ computational security parameter
  - $q > 2^\lambda$ prime
- Cyclic group $G \simeq G^q \times F$
  - $F = \langle f \rangle$ – subgroup of order q with easy DLog
  - $G^q = \langle g \rangle$ – subgroup of qth powers with unknown order
- Hardness assumptions
  - *ORD*: hard to find the order of any $h \in G \setminus F$
  - *HSM*: hard to distinguish random elements of G and G^q
- Advantages
  - can choose q freely as large prime
  - transparent setup
  - faster and smaller than Paillier ($\rightsquigarrow$ BICYCL by Bouvier et al. [BCIL22])

HSM-CL Linearly Homomorphic Encryption [CLT18; CCLST20]

$\underline{\mathsf{Setup}(1^\lambda, q)}$

1. Output $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$

HSM-CL Linearly Homomorphic Encryption [CLT18; CCLST20]

$\underline{\mathsf{Setup}(1^\lambda, q)}$

1. Output $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$

$\underline{\mathsf{KeyGen}(\mathsf{pp})}$

1. Sample $\mathsf{sk} \leftarrow_R [0, 2^{\mathsf{large}})$, set $\mathsf{pk} := g^{\mathsf{sk}}$
2. Output $(\mathsf{pk}, \mathsf{sk})$

HSM-CL Linearly Homomorphic Encryption [CLT18; CCLST20]

$\underline{\mathsf{Setup}(1^\lambda, q)}$

1. Output $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$

$\underline{\mathsf{KeyGen}(\mathsf{pp})}$

1. Sample $\mathsf{sk} \leftarrow_R [0, 2^{\mathsf{large}})$, set $\mathsf{pk} := g^{\mathsf{sk}}$
2. Output $(\mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{Enc}(\mathsf{pk}, m \in \mathbb{F}_q)}$

1. Sample $r \leftarrow_R [0, 2^{\mathsf{large}})$
2. Output $\mathsf{ct} := (g^r, f^m \cdot \mathsf{pk}^r)$

HSM-CL Linearly Homomorphic Encryption [CLT18; CCLST20]

$\underline{\mathsf{Setup}(1^\lambda, q)}$

1. Output $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$

$\underline{\mathsf{KeyGen}(\mathsf{pp})}$

1. Sample $\mathsf{sk} \leftarrow_R [0, 2^{\mathsf{large}})$, set $\mathsf{pk} := g^{\mathsf{sk}}$
2. Output $(\mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{Enc}(\mathsf{pk}, m \in \mathbb{F}_q)}$

1. Sample $r \leftarrow_R [0, 2^{\mathsf{large}})$
2. Output $\mathsf{ct} := (g^r, f^m \cdot \mathsf{pk}^r)$

$\underline{\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})}$

1. Compute $f^m := \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}}$
2. Output m

HSM-CL Linearly Homomorphic Encryption [CLT18; CCLST20]

$\underline{\mathsf{Setup}(1^\lambda, q)}$

1. Output $\mathsf{pp} \leftarrow \mathsf{CLGen}(1^\lambda, q)$

$\underline{\mathsf{Enc}(\mathsf{pk}, m \in \mathbb{F}_q)}$

1. Sample $r \leftarrow_R [0, 2^{\mathsf{large}})$
2. Output $\mathsf{ct} := (g^r, f^m \cdot \mathsf{pk}^r)$

$\underline{\mathsf{KeyGen}(\mathsf{pp})}$

1. Sample $\mathsf{sk} \leftarrow_R [0, 2^{\mathsf{large}})$, set $\mathsf{pk} := g^{\mathsf{sk}}$
2. Output $(\mathsf{pk}, \mathsf{sk})$

$\underline{\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})}$

1. Compute $f^m := \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}}$
2. Output m

- IND-CPA secure by the *HSM* assumption
- Analogue of Camenisch-Shoup encryption for the CL framework

The CDN Paradigm for MPC [CDN01]

Ingredients

- Threshold Linearly Homomorphic Encryption
- ZK Proof of Plaintext Knowledge (PoPK)
- ZK Proof of Correct Multiplication (PoCM)

The CDN Paradigm for MPC [CDN01]

Ingredients

- Threshold Linearly Homomorphic Encryption
- ZK Proof of Plaintext Knowledge (PoPK)
- ZK Proof of Correct Multiplication (PoCM)

Highlevel Overview

- Input: encrypt input + PoPK
- Output: threshold decryption
- Linear operations: use homomorphic properties

The CDN Paradigm for MPC [CDN01]

Ingredients

- Threshold Linearly Homomorphic Encryption
- ZK Proof of Plaintext Knowledge (PoPK)
- ZK Proof of Correct Multiplication (PoCM)

Highlevel Overview

- Input: encrypt input + PoPK
- Output: threshold decryption
- Linear operations: use homomorphic properties
- Multiplication $\mathsf{ct}_z \leftarrow \mathsf{ct}_x \cdot \mathsf{ct}_y$:
  1. jointly sample mask $\mathsf{ct}_d, [\![d]\!]$ such that $d \in_r \mathbb{F}_q$
  2. create additive sharing $[\![x]\!] \leftarrow [\![d]\!] - \mathsf{TDec}(\mathsf{ct}_x + \mathsf{ct}_d)$
  3. broadcast $\mathsf{ct}_{z_i} \leftarrow [\![x]\!]_i \cdot \mathsf{ct}_y$ with PoCM, and accumulate $\mathsf{ct}_z \leftarrow \sum_i \mathsf{ct}_{z_i}$

Setting

Security model

- active security
- static corruptions
- honest majority ($t < N/2$)
- broadcast available

Goals

- guaranteed output delivery
- transparent setup

Zero-Knowledge

Example: Schnorr Proof over $\mathbb{Z}$ – $\mathsf{R}_{\mathsf{DLog}} := \{h; x \mid h = g^x\}$

Prover	Verifier

Example: Schnorr Proof over $\mathbb{Z}$ – $\mathsf{R}_{\mathsf{DLog}} := \{h; x \mid h = g^x\}$

Prover		**Verifier**
$r \in_R [A],\ t := g^r$	$\xrightarrow{\quad t \quad}$	

Example: Schnorr Proof over $\mathbb{Z}$ – $\mathrm{R}_{\mathrm{DLog}} := \{h; x \mid h = g^x\}$

Prover		**Verifier**
$r \in_R [A],\ t := g^r$	$\xrightarrow{\quad t \quad}$	
	$\xleftarrow{\quad k \in_R [C] \quad}$	

Example: Schnorr Proof over $\mathbb{Z}$ – $\mathrm{R}_{\mathrm{DLog}} := \{h; x \mid h = g^x\}$

Prover		**Verifier**
$r \in_R [A],\ t := g^r$	$\xrightarrow{\quad t \quad}$	
	$\xleftarrow{\quad k \in_R [C] \quad}$	
$u \leftarrow r + k \cdot x \in \mathbb{Z}$	$\xrightarrow{\quad u \quad}$	Check: $g^u \stackrel{?}{=} t \cdot h^k$

Example: Schnorr Proof over $\mathbb{Z}$ – $\mathrm{R_{DLog}} := \{h; x \mid h = g^x\}$

$$
\begin{array}{lcl}
\textbf{Prover} & & \textbf{Verifier} \\
r \in_R [A],\ t := g^r & \xrightarrow{\quad t \quad} & \\
 & \xleftarrow{\quad k \in_R [C] \quad} & \\
u \leftarrow r + k \cdot x \in \mathbb{Z} & \xrightarrow{\quad u \quad} & \text{Check: } g^u \stackrel{?}{=} t \cdot h^k
\end{array}
$$

Knowledge Soundness: Extract from accepting $(t, k, u), (t, k', u')$ with $k \neq k'$:

$$x = (u - u') \cdot (k - k')^{-1} \pmod{\operatorname{ord}(g)}$$

Example: Schnorr Proof over $\mathbb{Z}$ – $\mathsf{R}_{\mathsf{DLog}} := \{h; x \mid h = g^x\}$

Prover		**Verifier**
$r \in_R [A],\ t := g^r$	$\xrightarrow{\quad t \quad}$	
	$\xleftarrow{\quad k \in_R [C] \quad}$	
$u \leftarrow r + k \cdot x \in \mathbb{Z}$	$\xrightarrow{\quad u \quad}$	Check: $g^u \stackrel{?}{=} t \cdot h^k$

Knowledge Soundness: Extract from accepting $(t, k, u), (t, k', u')$ with $k \neq k'$:

$$x = (u - u') \cdot (k - k')^{-1} \pmod{\mathrm{ord}(g)}$$

unknown order!

Example: Schnorr Proof over $\mathbb{Z}$ – $\mathrm{R}_{\mathrm{DLog}} := \{h; x \mid h = g^x\}$

Prover		**Verifier**
$r \in_R [A],\ t := g^r$	$\xrightarrow{\quad t \quad}$	
	$\xleftarrow{\quad k \in_R [C] \quad}$	
$u \leftarrow r + k \cdot x \in \mathbb{Z}$	$\xrightarrow{\quad u \quad}$	Check: $g^u \stackrel{?}{=} t \cdot h^k$

Knowledge Soundness: Extract from accepting $(t, k, u), (t, k', u')$ with $k \neq k'$:

$$x = (u - u') \cdot (k - k')^{-1} \pmod{\mathrm{ord}(g)}$$

Over the integers?

unknown order!

Example: Schnorr Proof over $\mathbb{Z}$ – $\mathrm{R}_{\mathrm{DLog}} := \{h; x \mid h = g^x\}$

Prover		**Verifier**
$r \in_R [A],\ t := g^r$	$\xrightarrow{\quad t \quad}$	
	$\xleftarrow{\quad k \in_R [C] \quad}$	
$u \leftarrow r + k \cdot x \in \mathbb{Z}$	$\xrightarrow{\quad u \quad}$	Check: $g^u \stackrel{?}{=} t \cdot h^k$

Knowledge Soundness: Extract from accepting $(t, k, u), (t, k', u')$ with $k \neq k'$:

$$x = (u - u') \cdot (k - k')^{-1} \pmod{\mathrm{ord}(g)}$$

unknown order!

Over the integers?

- Binary challenges $\rightsquigarrow$ repetitions

Example: Schnorr Proof over $\mathbb{Z}$ – $\mathsf{R}_{\mathsf{DLog}} := \{h; x \mid h = g^x\}$

Prover		**Verifier**
$r \in_R [A],\ t := g^r$	$\xrightarrow{\quad t \quad}$	
	$\xleftarrow{\quad k \in_R [C] \quad}$	
$u \leftarrow r + k \cdot x \in \mathbb{Z}$	$\xrightarrow{\quad u \quad}$	Check: $g^u \stackrel{?}{=} t \cdot h^k$

Knowledge Soundness: Extract from accepting $(t, k, u), (t, k', u')$ with $k \neq k'$:

$$x = (u - u') \cdot (k - k')^{-1} \pmod{\operatorname{ord}(g)}$$

unknown order!

Over the integers?

- Binary challenges $\rightsquigarrow$ repetitions
- Strong Root / Low Order assumptions $\rightsquigarrow$ additional setup and complications

Example: Schnorr Proof over $\mathbb{Z}$ – $R_{\mathsf{DLog}} := \{h; x \mid h = g^x\}$

Prover		**Verifier**
$r \in_R [A],\ t := g^r$	$\xrightarrow{\quad t \quad}$	
	$\xleftarrow{\quad k \in_R [C] \quad}$	
$u \leftarrow r + k \cdot x \in \mathbb{Z}$	$\xrightarrow{\quad u \quad}$	Check: $g^u \stackrel{?}{=} t \cdot h^k$

Knowledge Soundness: Extract from accepting $(t, k, u), (t, k', u')$ with $k \neq k'$:

$$x = (u - u') \cdot (k - k')^{-1} \pmod{\mathsf{ord}(g)}$$

unknown order!

Over the integers?

- Binary challenges $\rightsquigarrow$ repetitions
- Strong Root / Low Order assumptions $\rightsquigarrow$ additional setup and complications
- Sometimes normal, set-membership soundness ($\exists x \,.\, h = g^x$) is enough!

New Assumption

Definition (C-Rough Order Assumption (informal))

Let $C \in \mathbb{N}$. The following are computationally indistinguishable:

New Assumption

Definition (C-Rough Order Assumption (informal))

Let $C \in \mathbb{N}$. The following are computationally indistinguishable:

1. class groups generated by CLGen

New Assumption

Definition (C-Rough Order Assumption (informal))

Let $C \in \mathbb{N}$. The following are computationally indistinguishable:

1. class groups generated by CLGen
2. class groups generated by CLGen with a C-rough order ($\mathrm{ord}(G)$ has no divisors $< C$)

New Assumption

Definition (C-Rough Order Assumption (informal))

Let $C \in \mathbb{N}$. The following are computationally indistinguishable:

1. class groups generated by CLGen
2. class groups generated by CLGen with a C-rough order ($\mathrm{ord}(G)$ has no divisors $< C$)

How does it help?

- C-rough order $\implies$ all $x \in [1, C)$ are invertible modulo $\mathrm{ord}(G)$
  $\implies (k - k')^{-1}$ exists $\implies$ witness exists

New Assumption

Definition (C-Rough Order Assumption (informal))

Let $C \in \mathbb{N}$. The following are computationally indistinguishable:

1. class groups generated by CLGen
2. class groups generated by CLGen with a C-rough order ($\mathrm{ord}(G)$ has no divisors $< C$)

How does it help?

- C-rough order $\implies$ all $x \in [1, C)$ are invertible modulo $\mathrm{ord}(G)$
  $\implies$ $(k - k')^{-1}$ exists $\implies$ witness exists

Justified?

- Cohen-Lenstra heuristic [CL84] $\rightsquigarrow$ class group orders roughly "behave like random integers"
  $\implies$ there are significantly many C-rough-order class groups
- Efficient distinguisher would be great!

Building Threshold Encryption

Our Goal: $\mathcal{F}_{\mathsf{TE}}$ Ideal Functionality

$\mathcal{F}_{\mathsf{TE}}$

Key Generation

- Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
- Output pk to all parties and store sk

Our Goal: $\mathcal{F}_{\mathsf{TE}}$ Ideal Functionality

$\mathcal{F}_{\mathsf{TE}}$

Key Generation

- Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
- Output pk to all parties and store sk

Threshold Decryption

- On input $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ from at least $t + 1$ parties, compute $M := \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}}$
- Output $m := \log_f(M)$ to all parties

Our Goal: $\mathcal{F}_{\mathsf{TE}}$ Ideal Functionality

$\mathcal{F}_{\mathsf{TE}}$

Key Generation

- Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
- Output pk to all parties and store sk

Threshold Decryption

- On input $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ from at least $t + 1$ parties, compute $M := \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}}$
- Output $m := \log_f(M)$ to all parties

Our Goal: $\mathcal{F}_{\mathsf{TE}}$ Ideal Functionality

$\mathcal{F}_{\mathsf{TE}}$

Key Generation

- Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
- Output pk to all parties and store sk

$\implies$ (t, N)-threshold secret sharing

Threshold Decryption

- On input $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ from at least $t+1$ parties, compute $M := \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}}$
- Output $m := \log_f(M)$ to all parties

Our Goal: $\mathcal{F}_{\mathsf{TE}}$ Ideal Functionality

$\mathcal{F}_{\mathsf{TE}}$

Key Generation

- Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$
- Output pk to all parties and store sk

$\Longrightarrow$ (t, N)-threshold secret sharing

Threshold Decryption

- On input $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ from at least $t+1$ parties, compute $M := \mathsf{ct}_2 \cdot \mathsf{ct}_1^{-\mathsf{sk}}$
- Output $m := \log_f(M)$ to all parties

reconstruction in the exponent of unknown order group element

Pedersen-style Distributed Key Generation

1. All parties P_i
   1.1 sample contribution α_i
   1.2 publish g^{α_i}
   1.3 share $\alpha_i \rightarrow \langle \alpha_i \rangle$

Pedersen-style Distributed Key Generation

1. All parties P_i
   1.1 sample contribution α_i
   1.2 publish g^{α_i}
   1.3 share $\alpha_i \to \langle \alpha_i \rangle$

3. Define public key $\mathsf{pk} := \prod_{P_i} g^{\alpha_i}$
4. Have shared secret key $\langle \mathsf{sk} \rangle := \sum_{P_i} \langle \alpha_i \rangle$

Pedersen-style Distributed Key Generation

1. All parties P_i
   1.1 sample contribution α_i
   1.2 publish g^{α_i}
   1.3 share $\alpha_i \to \langle \alpha_i \rangle$

3. Define public key $\mathsf{pk} := \prod_{P_i} g^{\alpha_i}$
4. Have shared secret key $\langle \mathsf{sk} \rangle := \sum_{P_i} \langle \alpha_i \rangle$

Shamir's Secret Sharing over $\mathbb{Z}$

Sharing

Just do it over the integers: To share $\alpha_i \in [0, 2^\ell)$,

- sample random $f(X) := \alpha_i + \sum_{k=1}^{t} r_k \cdot X^k$ with large enough r_k
- give $y_j := f(j)$ to P_j

$\implies$ See 90's papers for threshold RSA [DF92; FGMY97; Rab98]

Shamir's Secret Sharing over $\mathbb{Z}$

Sharing

Just do it over the integers: To share $\alpha_i \in [0, 2^\ell)$,

- sample random $f(X) := \alpha_i + \sum_{k=1}^{t} r_k \cdot X^k$ with large enough r_k
- give $y_j := f(j)$ to P_j

⚠ $f(j) \bmod j$ leaks $\alpha_i \bmod j$

$\implies$ See 90's papers for threshold RSA [DF92; FGMY97; Rab98]

Shamir's Secret Sharing over $\mathbb{Z}$

Sharing

Just do it over the integers: To share $\alpha_i \in [0, 2^\ell)$,

- sample random $f(X) := \alpha_i + \sum_{k=1}^{t} r_k \cdot X^k$ with large enough r_k
- give $y_j := f(j)$ to P_j

⚠ $f(j) \bmod j$ leaks $\alpha_i \bmod j$

Define $\Delta := N!$

$\implies$ See 90's papers for threshold RSA [DF92; FGMY97; Rab98]

Sharing

Just do it over the integers: To share $\alpha_i \in [0, 2^\ell)$,

- sample random $f(X) := \alpha_i \cdot \Delta + \sum_{k=1}^{t} r_k \cdot X^k$ with large enough r_k
- give $y_j := f(j)$ to P_j

Define $\Delta := N!$

⚠ $f(j) \bmod j$ leaks $\alpha_i \bmod j$

share $\alpha_i \cdot \Delta$ instead ✓

$\Longrightarrow$ See 90's papers for threshold RSA [DF92; FGMY97; Rab98]

Shamir's Secret Sharing over $\mathbb{Z}$

Sharing

Just do it over the integers: To share $\alpha_i \in [0, 2^\ell)$,

- sample random $f(X) := \alpha_i \cdot \Delta + \sum_{k=1}^{t} r_k \cdot X^k$ with large enough r_k
- give $y_j := f(j)$ to P_j

Define $\Delta := N!$

Reconstruction (in the Exponent)

Lagrange interpolation: Given $\geq t+1$ shares $(x_j = j, y_j = f(j))$, compute

$$f(X) = \sum_i y_i \cdot \prod_{j \neq i} \frac{x_j - X}{x_j - x_i}$$

$\Longrightarrow$ See 90's papers for threshold RSA [DF92; FGMY97; Rab98]

Shamir's Secret Sharing over $\mathbb{Z}$

Sharing

Just do it over the integers: To share $\alpha_i \in [0, 2^\ell)$,

- sample random $f(X) := \alpha_i \cdot \Delta + \sum_{k=1}^{t} r_k \cdot X^k$ with large enough r_k
- give $y_j := f(j)$ to P_j

Define $\Delta := N!$

Reconstruction (in the Exponent)

Lagrange interpolation: Given $\geq t+1$ shares $(x_j = j, y_j = f(j))$, compute

$$f(X) = \sum_i y_i \cdot \prod_{j \neq i} \frac{x_j - X}{x_j - x_i}$$

unknown group order $\implies$ cannot divide in the exponent

$\implies$ See 90's papers for threshold RSA [DF92; FGMY97; Rab98]

Shamir's Secret Sharing over $\mathbb{Z}$

Sharing

Just do it over the integers: To share $\alpha_i \in [0, 2^\ell)$,

- sample random $f(X) := \alpha_i \cdot \Delta + \sum_{k=1}^{t} r_k \cdot X^k$ with large enough r_k
- give $y_j := f(j)$ to P_j

Define $\Delta := N!$

Reconstruction (in the Exponent)

Lagrange interpolation: Given $\geq t+1$ shares $(x_j = j, y_j = f(j))$, compute

$$f(X) = \sum_i y_i \cdot \prod_{j \neq i} \frac{x_j - X}{x_j - x_i} \cdot \Delta$$

⚠ unknown group order $\implies$ cannot divide in the exponent

multiply by Δ, reconstruct $\alpha_i \cdot \Delta^2$ ✓

$\implies$ See 90's papers for threshold RSA [DF92; FGMY97; Rab98]

Pedersen-style Distributed Key Generation

1. All parties P_i
   1.1 sample contribution α_i
   1.2 publish g^{α_i}
   1.3 share $\alpha_i \rightarrow \langle \alpha_i \rangle$

3. Define public key $\mathsf{pk} := \prod_{P_i} g^{\alpha_i}$
4. Have shared secret key $\langle \mathsf{sk} \rangle := \sum_{P_i} \langle \alpha_i \rangle$

Pedersen-style Distributed Key Generation

1. All parties P_i
   1.1 sample contribution α_i
   1.2 publish g^{α_i}
   1.3 share $\alpha_i \to \langle \alpha_i \rangle$
   1.4 prove consistency

3. Define public key $\mathsf{pk} := \prod_{P_i} g^{\alpha_i}$
4. Have shared secret key $\langle \mathsf{sk} \rangle := \sum_{P_i} \langle \alpha_i \rangle$

Feldman's Verifiable Secret Sharing over $\mathbb{Z}$

Goal: Shares of α_i consistent with each other and g^{α_i}

Recall: Sharing polynomial $f(X) := \alpha_i \cdot \Delta + \sum_{k=1}^{t} r_k \cdot X^k$ and shares $(x_j = j, y_j = f(j))$

Feldman's Verifiable Secret Sharing over $\mathbb{Z}$

Goal: Shares of α_i consistent with each other and g^{α_i}

Recall: Sharing polynomial $f(X) := \alpha_i \cdot \Delta + \sum_{k=1}^{t} r_k \cdot X^k$ and shares $(x_j = j, y_j = f(j))$

F-Share:

- additionally publish $C_0 := g^{\alpha_i}$ and $C_k := g^{\Delta \cdot r_k}$ for $k \in [1, t]$
- prove that $C_k \in \langle g \rangle$

commit to coefficients of f

Feldman's Verifiable Secret Sharing over $\mathbb{Z}$

Goal: Shares of α_i consistent with each other and g^{α_i}

Recall: Sharing polynomial $f(X) := \alpha_i \cdot \Delta + \sum_{k=1}^{t} r_k \cdot X^k$ and shares $(x_j = j, y_j = f(j))$

F-Share:

- additionally publish $C_0 := g^{\alpha_i}$ and $C_k := g^{\Delta \cdot r_k}$ for $k \in [1, t]$
- prove that $C_k \in \langle g \rangle$

F-Check: $P_j \neq P_i$ checks

$$g^{\Delta \cdot y_j} \stackrel{?}{=} C_0^{\Delta^2} \cdot \prod_{k=1}^{t} (C_k)^{(j^k)} = g^{\overbrace{\Delta^2 \cdot \alpha_i + \sum_{k=1}^{t} \Delta \cdot r_k \cdot j^k}^{\text{evaluate } \Delta \cdot f(j) \text{ in the exponent}}}$$

Feldman's Verifiable Secret Sharing over $\mathbb{Z}$

Goal: Shares of α_i consistent with each other and g^{α_i}

Recall: Sharing polynomial $f(X) := \alpha_i \cdot \Delta + \sum_{k=1}^{t} r_k \cdot X^k$ and shares $(x_j = j, y_j = f(j))$

F-Share:

- additionally publish $C_0 := g^{\alpha_i}$ and $C_k := g^{\Delta \cdot r_k}$ for $k \in [1, t]$
- prove that $C_k \in \langle g \rangle$

F-Check: $P_j \neq P_i$ checks

$$g^{\Delta \cdot y_j} \stackrel{?}{=} C_0^{\Delta^2} \cdot \prod_{k=1}^{t} (C_k)^{(j^k)} = g^{\Delta^2 \cdot \alpha_i + \sum_{k=1}^{t} \Delta \cdot r_k \cdot j^k}$$

ORD Assumption $\wedge$ $\gcd(\mathrm{ord}(g), \Delta) = 1$ $\wedge$ PoK for $C_0 = g^{\alpha_i}$ $\implies$ Integer VSS

Feldman's Verifiable Secret Sharing over $\mathbb{Z}$

Goal: Shares of α_i consistent with each other and g^{α_i}

Recall: Sharing polynomial $f(X) := \alpha_i \cdot \Delta + \sum_{k=1}^{t} r_k \cdot X^k$ and shares $(x_j = j, y_j = f(j))$

F-Share:

- additionally publish $C_0 := g^{\alpha_i}$ and $C_k := g^{\Delta \cdot r_k}$ for $k \in [1, t]$
- prove that $C_k \in \langle g \rangle$

F-Check: $P_j \neq P_i$ checks

$$g^{\Delta \cdot y_j} \stackrel{?}{=} C_0^{\Delta^2} \cdot \prod_{k=1}^{t} (C_k)^{(j^k)} = g^{\Delta^2 \cdot \alpha_i + \sum_{k=1}^{t} \Delta \cdot r_k \cdot j^k}$$

ORD Assumption $\wedge$ $\gcd(\mathrm{ord}(g), \Delta) = 1$ $\wedge$ PoK for $C_0 = g^{\alpha_i}$ $\Longrightarrow$ Integer VSS

- Issues with Rabin's VSS [Rab98]: Does not use *ORD*
  $\Longrightarrow$ Corrupt dealer knowing $\mathrm{ord}(g)$ can prevent reconstruction

Pedersen-style Distributed Key Generation

1. All parties P_i
   1.1 sample contribution α_i
   1.2 publish g^{α_i}
   1.3 share $\alpha_i \rightarrow \langle \alpha_i \rangle$
   1.4 prove consistency

3. Define public key $\mathsf{pk} := \prod_{P_i} g^{\alpha_i}$
4. Have shared secret key $\langle \mathsf{sk} \rangle := \sum_{P_i} \langle \alpha_i \rangle$

Pedersen-style Distributed Key Generation

1. All parties P_i
   1.1 sample contribution α_i
   1.2 publish g^{α_i}
   1.3 share $\alpha_i \to \langle \alpha_i \rangle$
   1.4 prove consistency
2. Disqualify misbehaving parties $\rightsquigarrow$ set of $\geq t+1$ remaining parties $\mathcal{Q}$
3. Define public key $\mathsf{pk} := \prod_{P_i \in \mathcal{Q}} g^{\alpha_i}$
4. Have shared secret key $\langle \mathsf{sk} \rangle := \sum_{P_i \in \mathcal{Q}} \langle \alpha_i \rangle$

Pedersen-style Distributed Key Generation

1. All parties P_i
   1.1 sample contribution α_i
   1.2 publish g^{α_i}
   1.3 share $\alpha_i \to \langle \alpha_i \rangle$
   1.4 prove consistency
2. Disqualify misbehaving parties $\rightsquigarrow$ set of $\geq t+1$ remaining parties $\mathcal{Q}$
3. Define public key $\boxed{\mathsf{pk} := \prod_{P_i \in \mathcal{Q}} g^{\alpha_i}}$
4. Have shared secret key $\langle \mathsf{sk} \rangle := \sum_{P_i \in \mathcal{Q}} \langle \alpha_i \rangle$

Pedersen-style Distributed Key Generation

1. All parties P_i
   1.1 sample contribution α_i
   1.2 publish g^{α_i}
   1.3 share $\alpha_i \to \langle \alpha_i \rangle$
   1.4 prove consistency
2. Disqualify misbehaving parties $\rightsquigarrow$ set of $\geq t+1$ remaining parties $\mathcal{Q}$
3. Define public key $\boxed{\mathsf{pk} := \prod_{P_i \in \mathcal{Q}} g^{\alpha_i}}$
4. Have shared secret key $\langle \mathsf{sk} \rangle := \sum_{P_i \in \mathcal{Q}} \langle \alpha_i \rangle$

⚠ $\mathcal{A}$ can bias distribution of pk!
(Gennaro et al. [GJKR07])

Fixing the Bias

Gennaro et al. [GJKR07]: Unbiased DKG with two-stage approach + Pedersen VSS

- needs additional rounds and extra setup (but bias is ok for e.g. Schnorr signatures)

~~Fixing~~ Living with the Bias

Gennaro et al. [GJKR07]: Unbiased DKG with two-stage approach + Pedersen VSS

- needs additional rounds and extra setup (but bias is ok for e.g. Schnorr signatures)

Allowing the Adversary to bias the distribution:

Similar observations for ElGamal by Courtier et al. [CGGI13] and Stengele et al. [SRMH21].

~~Fixing~~ Living with the Bias

Gennaro et al. [GJKR07]: Unbiased DKG with two-stage approach + Pedersen VSS

- needs additional rounds and extra setup (but bias is ok for e.g. Schnorr signatures)

Allowing the Adversary to bias the distribution:

$\underline{\mathsf{BiasedKeyGen}^{\mathcal{A}}}$

1. $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}$

Similar observations for ElGamal by Courtier et al. [CGGI13] and Stengele et al. [SRMH21].

~~Fixing~~ Living with the Bias

Gennaro et al. [GJKR07]: Unbiased DKG with two-stage approach + Pedersen VSS

- needs additional rounds and extra setup (but bias is ok for e.g. Schnorr signatures)

Allowing the Adversary to bias the distribution:

$\underline{\mathsf{BiasedKeyGen}^{\mathcal{A}}}$

1. $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}$
2. $\boxed{\delta \leftarrow \mathcal{A}(\mathsf{pk}^*)}$

Similar observations for ElGamal by Courtier et al. [CGGI13] and Stengele et al. [SRMH21].

~~Fixing~~ Living with the Bias

Gennaro et al. [GJKR07]: Unbiased DKG with two-stage approach + Pedersen VSS

- needs additional rounds and extra setup (but bias is ok for e.g. Schnorr signatures)

Allowing the Adversary to bias the distribution:

$\underline{\mathsf{BiasedKeyGen}^{\mathcal{A}}}$

1. $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}$
2. $\boxed{\delta \leftarrow \mathcal{A}(\mathsf{pk}^*)}$
3. Output $\mathsf{sk} := \mathsf{sk}^* + \boxed{\delta}$,

   $\mathsf{pk} := g^{\mathsf{sk}} = \mathsf{pk}^* \cdot g^{\boxed{\delta}}$

Similar observations for ElGamal by Courtier et al. [CGGI13] and Stengele et al. [SRMH21].

~~Fixing~~ Living with the Bias

Gennaro et al. [GJKR07]: Unbiased DKG with two-stage approach + Pedersen VSS

- needs additional rounds and extra setup (but bias is ok for e.g. Schnorr signatures)

Allowing the Adversary to bias the distribution:

IND-CPA by reduction of unbiased encryption:

<u>$\mathsf{BiasedKeyGen}^{\mathcal{A}}$</u>

1. $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}$
2. $\delta \leftarrow \mathcal{A}(\mathsf{pk}^*)$
3. Output $\mathsf{sk} := \mathsf{sk}^* + \delta$,
   $\mathsf{pk} := g^{\mathsf{sk}} = \mathsf{pk}^* \cdot g^{\delta}$

Similar observations for ElGamal by Courtier et al. [CGGI13] and Stengele et al. [SRMH21].

~~Fixing~~ Living with the Bias

Gennaro et al. [GJKR07]: Unbiased DKG with two-stage approach + Pedersen VSS

- needs additional rounds and extra setup (but bias is ok for e.g. Schnorr signatures)

Allowing the Adversary to bias the distribution:

IND-CPA by reduction of unbiased encryption:

1. given encryption under (unbiased) pk^*

$$\mathsf{ct} := (g^r, (\mathsf{pk}^*)^r \cdot f^m)$$

$\underline{\mathsf{BiasedKeyGen}^{\mathcal{A}}}$

1. $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}$
2. $\boxed{\delta \leftarrow \mathcal{A}(\mathsf{pk}^*)}$
3. Output $\mathsf{sk} := \mathsf{sk}^* + \boxed{\delta}$,

   $\mathsf{pk} := g^{\mathsf{sk}} = \mathsf{pk}^* \cdot g^{\boxed{\delta}}$

Similar observations for ElGamal by Courtier et al. [CGGI13] and Stengele et al. [SRMH21].

~~Fixing~~ Living with the Bias

Gennaro et al. [GJKR07]: Unbiased DKG with two-stage approach + Pedersen VSS

- needs additional rounds and extra setup (but bias is ok for e.g. Schnorr signatures)

Allowing the Adversary to bias the distribution:

$\underline{\mathsf{BiasedKeyGen}^{\mathcal{A}}}$

1. $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}$
2. $\boxed{\delta \leftarrow \mathcal{A}(\mathsf{pk}^*)}$
3. Output $\mathsf{sk} := \mathsf{sk}^* + \boxed{\delta}$,

   $\mathsf{pk} := g^{\mathsf{sk}} = \mathsf{pk}^* \cdot g^{\boxed{\delta}}$

IND-CPA by reduction of unbiased encryption:

1. given encryption under (unbiased) pk^*

$$\mathsf{ct} := (g^r, (\mathsf{pk}^*)^r \cdot f^m)$$

2. compute encryption under (biased) pk

$$\begin{aligned} \mathsf{ct}' &:= (g^r, ((\mathsf{pk}^*)^r \cdot f^m) \cdot (g^r)^{\boxed{\delta}}) \\ &= (g^r, (\mathsf{pk})^r \cdot f^m) \end{aligned}$$

Similar observations for ElGamal by Courtier et al. [CGGI13] and Stengele et al. [SRMH21].

YOSO

YOSO MPC – You Only Speak Once

YOSO???

- large scale MPC for many parties
- work done by many small committees
- mechanism for passing secrets to future committees without knowing them
- each party sends only <u>one</u> round of messages

YOSO MPC – You Only Speak Once

YOSO???

- large scale MPC for many parties
- work done by many small committees
- mechanism for passing secrets to future committees without knowing them
- each party sends only <u>one</u> round of messages

Why is our work YOSO-friendly?

- transparent setup! – open problem in previous work [Gen+21]
- simple one-round distributed key generation and decryption protocols
- small secret state: only shared sk needs to be passed between the committees

Summary

Summary

Contributions

- First actively-secure threshold version of the HSM-CL cryptosystem
- UC-secure MPC using the CDN paradigm
- New zero-knowledge protocols for multiplicative relations of encrypted values
- Adaption to the YOSO setting and solution to the open problem of transparent setup

Summary

Contributions

- First actively-secure threshold version of the HSM-CL cryptosystem
- UC-secure MPC using the CDN paradigm
- New zero-knowledge protocols for multiplicative relations of encrypted values
- Adaption to the YOSO setting and solution to the open problem of transparent setup

Open problems

- [CLT22] give a HSM-CL (threshold) variant for $\mathbb{Z}_{2^k}$. Can we adapt our techniques?
- Threshold-friendly CCA-secure variant of the encryption scheme without a random oracle? ($\rightsquigarrow$ Cramer-Shoup-style)

Summary

Contributions

- First actively-secure threshold version of the HSM-CL cryptosystem
- UC-secure MPC using the CDN paradigm
- New zero-knowledge protocols for multiplicative relations of encrypted values
- Adaption to the YOSO setting and solution to the open problem of transparent setup

Open problems

- [CLT22] give a HSM-CL (threshold) variant for $\mathbb{Z}_{2^k}$. Can we adapt our techniques?
- Threshold-friendly CCA-secure variant of the encryption scheme without a random oracle? ($\rightsquigarrow$ Cramer-Shoup-style)

Full version on ePrint: `https://ia.cr/2022/1437`

Summary

Contributions

- First actively-secure threshold version of the HSM-CL cryptosystem
- UC-secure MPC using the CDN paradigm
- New zero-knowledge protocols for multiplicative relations of encrypted values
- Adaption to the YOSO setting and solution to the open problem of transparent setup

Open problems

- [CLT22] give a HSM-CL (threshold) variant for $\mathbb{Z}_{2^k}$. Can we adapt our techniques?
- Threshold-friendly CCA-secure variant of the encryption scheme without a random oracle? ($\rightsquigarrow$ Cramer-Shoup-style)

Full version on ePrint: `https://ia.cr/2022/1437`

Thank you!

References i

[BCIL22] C. Bouvier, G. Castagnos, L. Imbert, and F. Laguillaumie. **I want to ride my BICYCL: BICYCL Implements CryptographY in CLass groups.** Cryptology ePrint Archive, Report 2022/1466. `https://eprint.iacr.org/2022/1466`. 2022.

[CCLST20] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker. **“Bandwidth-Efficient Threshold EC-DSA”.** In: PKC 2020, Part II. May 2020.

[CDN01] R. Cramer, I. Damgård, and J. B. Nielsen. **“Multiparty Computation from Threshold Homomorphic Encryption”.** In: EUROCRYPT 2001. May 2001.

[CGGI13] V. Cortier, D. Galindo, S. Glondu, and M. Izabachène. **“Distributed ElGamal à la Pedersen: Application to Helios”.** In: Workshop on Privacy in the Electronic Society – WPES 2013. Nov. 2013.

[CL15] G. Castagnos and F. Laguillaumie. **"Linearly Homomorphic Encryption from DDH".** In: CT-RSA 2015. Apr. 2015.

[CL84] H. Cohen and H. W. Lenstra. **"Heuristics on class groups of number fields".** In: Number Theory Noordwijkerhout 1983. 1984.

[CLT18] G. Castagnos, F. Laguillaumie, and I. Tucker. **"Practical Fully Secure Unrestricted Inner Product Functional Encryption Modulo p".** In: ASIACRYPT 2018, Part II. Dec. 2018.

[CLT22] G. Castagnos, F. Laguillaumie, and I. Tucker. **"Threshold Linearly Homomorphic Encryption on $Z/2^kZ$".** In: ASIACRYPT 2022, Part II. Dec. 2022.

[DF92] Y. Desmedt and Y. Frankel. **"Shared Generation of Authenticators and Signatures (Extended Abstract)".** In: CRYPTO'91. Aug. 1992.

[FGMY97] Y. Frankel, P. Gemmell, P. D. MacKenzie, and M. Yung. **"Optimal Resilience Proactive Public-Key Cryptosystems".** In: 38th FOCS. Oct. 1997.

[Gen+21] C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov. **"YOSO: You Only Speak Once - Secure MPC with Stateless Ephemeral Roles".** In: CRYPTO 2021, Part II. Aug. 2021.

[GJKR07] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. **"Secure Distributed Key Generation for Discrete-Log Based Cryptosystems".** In: Journal of Cryptology 1 (Jan. 2007).

[Rab98] T. Rabin. **"A Simplified Approach to Threshold and Proactive RSA".** In: CRYPTO'98. Aug. 1998.

[SRMH21] O. Stengele, M. Raiber, J. Müller-Quade, and H. Hartenstein. **"ETHTID: Deployable Threshold Information Disclosure on Ethereum".** In: Conference on Blockchain Computing and Applications – BCCA 2021. Nov. 2021.