Threshold Cryptosystems from Threshold Fully Homomorphic Encryption

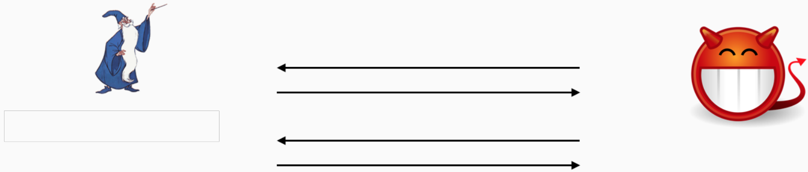
Sam Kim

Stanford University

Joint work with Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Peter M. R. Rasmussen, and Amit Sahai

Key Management

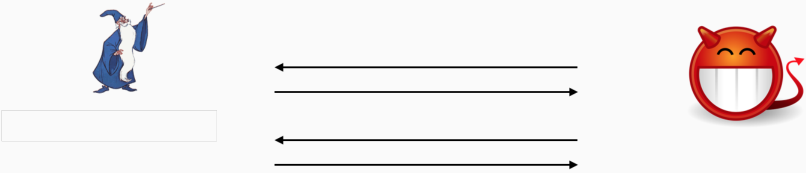
Cryptography is about using **hard problems** to our **advantage**.



Key Management

Cryptography is about using hard problems to our advantage.

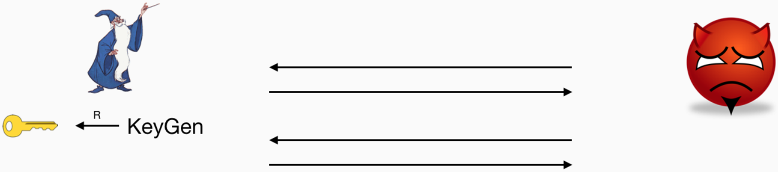
Crucial to have some sort of **secret information**.



Key Management

Cryptography is about using hard problems to our advantage.

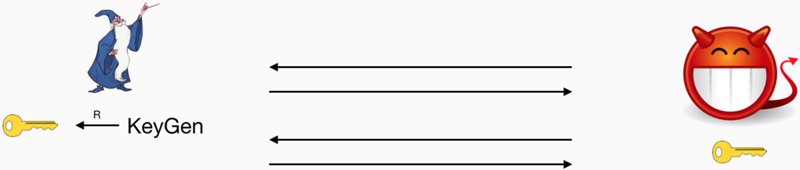
Crucial to have some sort of secret information.



Key Management

Cryptography is about using hard problems to our advantage.

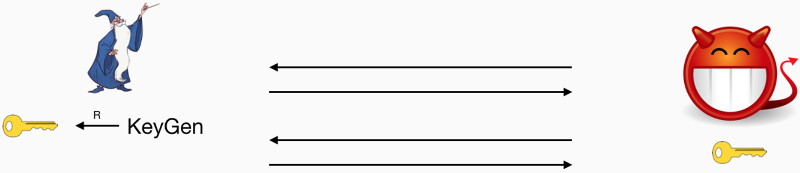
Crucial to have some sort of secret information.



Key Management

Cryptography is about using hard problems to our advantage.

Crucial to have some sort of secret information.



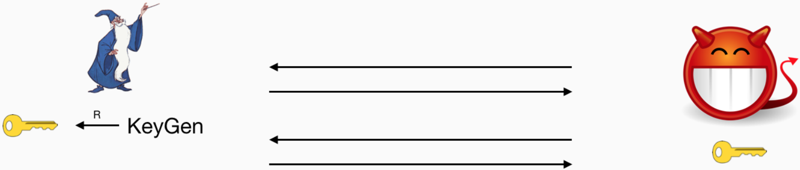
Key management is **hard** in practice!

(difficult to implement to crypto, systems get hacked, human error, ...)

Key Management

Cryptography is about using hard problems to our advantage.

Crucial to have some sort of secret information.

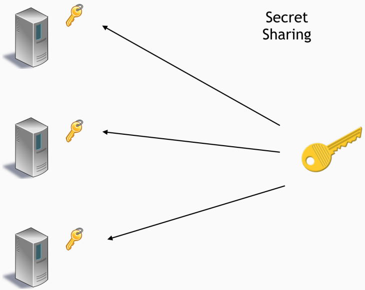


Key management is **hard** in practice!
(difficult to implement to crypto, systems get hacked, human error, ...)

Can we address key management at a more **fundamental** level?

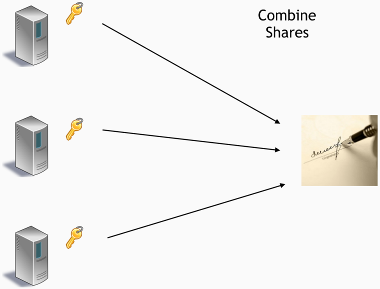
Threshold Cryptography

Can we divide the key k into shares s_1, ..., s_N and store them separately?



Threshold Cryptography

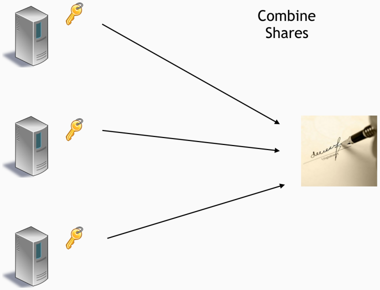
Can we divide the key k into shares s_1, ..., s_N and store them separately?



Correctness: Each server can *independently compute* on its key share $f(s_i, \cdot)$, which can later be *publicly combined* to form the final computation on the key $f(k, \cdot)$.

Threshold Cryptography

Can we divide the key k into shares s_1, ..., s_N and store them separately?

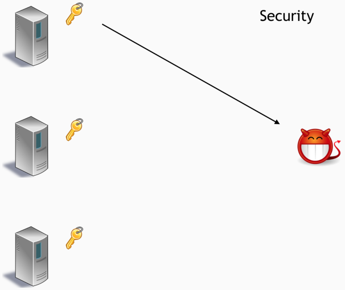


Correctness: Each server can *independently compute* on its key share $f(s_i, \cdot)$, which can later be *publicly combined* to form the final computation on the key $f(k, \cdot)$.

Security: *Hard* to form final computation $f(k, \cdot)$ without *all* the key shares $s_1, \ldots, s_N$.

Threshold Cryptography

Can we divide the key k into shares s_1, ..., s_N and store them separately?

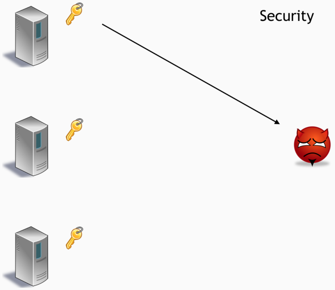


Correctness: Each server can *independently compute* on its key share $f(s_i, \cdot)$, which can later be *publicly combined* to form the final computation on the key $f(k, \cdot)$.

Security: *Hard* to form final computation $f(k, \cdot)$ without *all* the key shares $s_1, \ldots, s_N$.

Threshold Cryptography

Can we divide the key k into shares s_1, ..., s_N and store them separately?

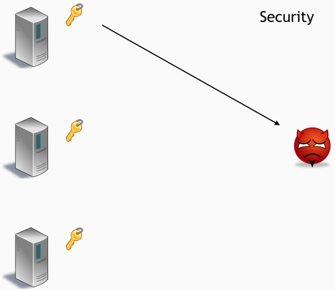


Correctness: Each server can *independently compute* on its key share $f(s_i, \cdot)$, which can later be *publicly combined* to form the final computation on the key $f(k, \cdot)$.

Security: *Hard* to form final computation $f(k, \cdot)$ without *all* the key shares $s_1, \ldots, s_N$.

Threshold Cryptography (t-out-of-N)

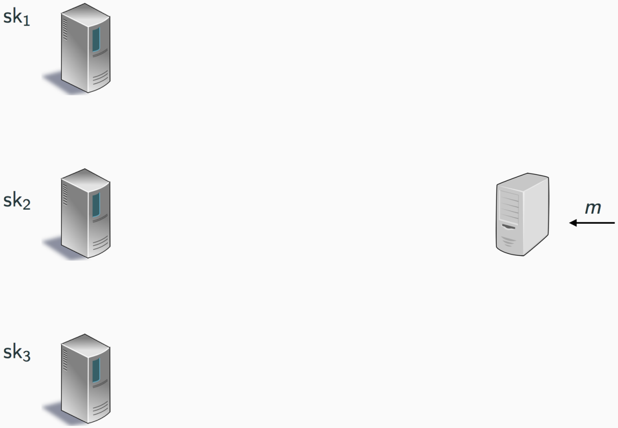
Can we divide the key k into shares s_1, ..., s_N and store them separately?



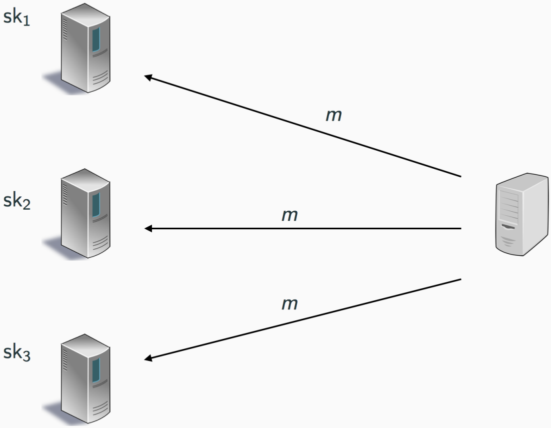
Correctness: Each server can independently compute on its key share $f(s_i, \cdot)$. Any **t** evaluation shares $f(s_i, \cdot)$ can be publicly combined to form final computation on the key $f(k, \cdot)$.

Security: Hard to form final computation $f(k, \cdot)$ without **t** key shares.

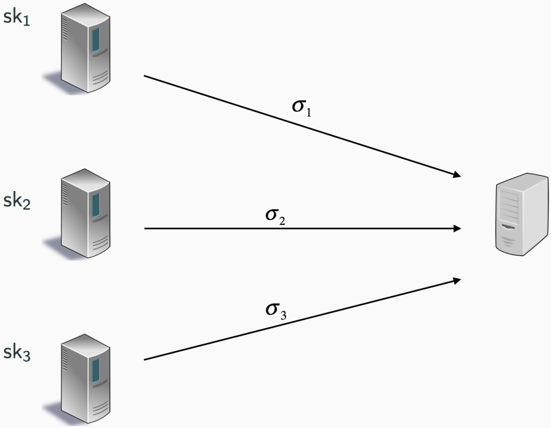
Threshold Signatures



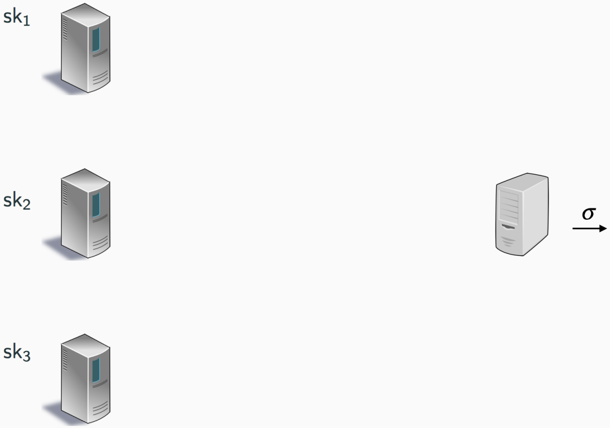
Threshold Signatures



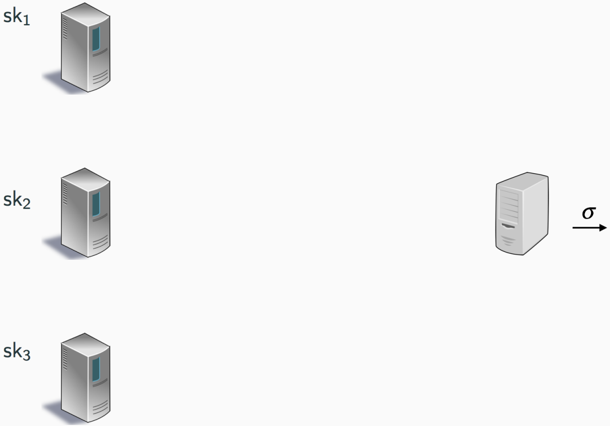
Threshold Signatures



Threshold Signatures

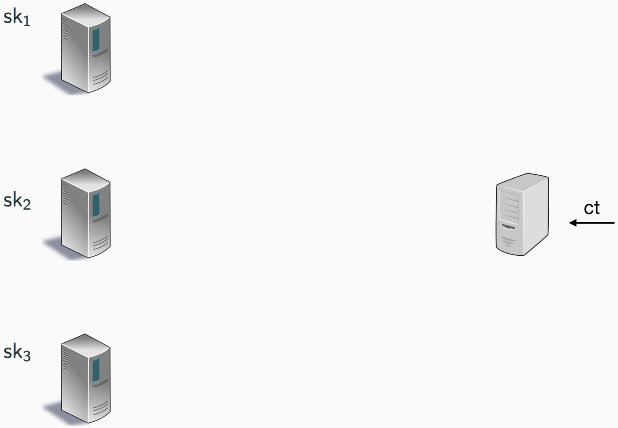


Threshold Signatures

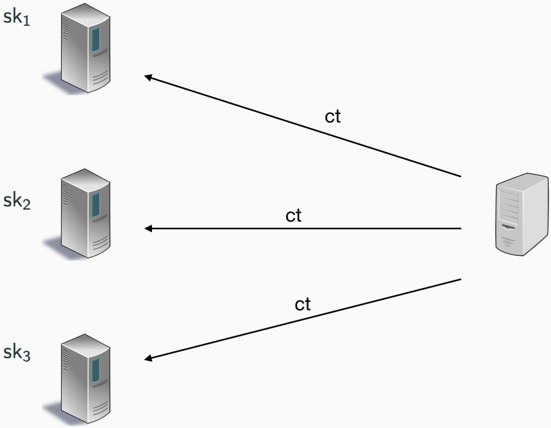


Requirements: Compactness, Correctness, Unforgeability, Anonymity, Robustness, ...

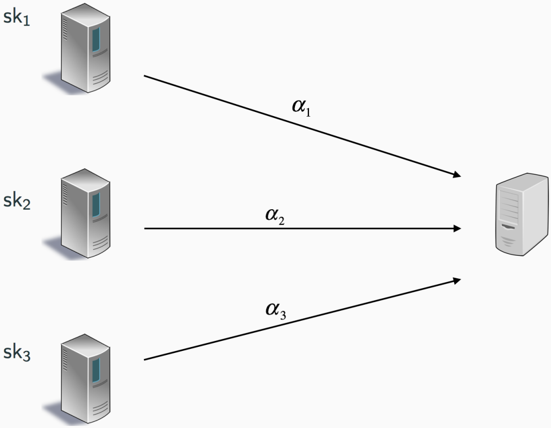
Threshold Public Key Encryption



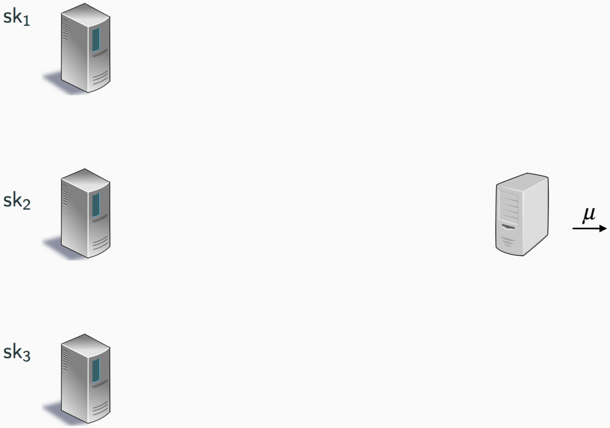
Threshold Public Key Encryption



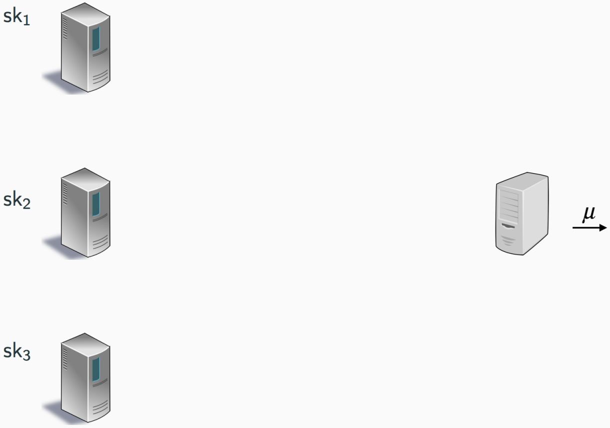
Threshold Public Key Encryption



Threshold Public Key Encryption



Threshold Public Key Encryption



Requirements: Compactness, Correctness, CCA security, Robustness, ...

Works on Threshold Cryptography

- RSA signatures [Fra89, DDFY94, GRJK07, Sho00]
- Schnorr signatures [SS01]
- (EC)DSA signatures [GJKR01, GGN16]
- BLS signatures [BLS04, Bol03]
- Cramer-Shoup encryption [CG99]
- Many more [SG02, DK05, BBH06, ...]

Works on Threshold Cryptography

- RSA signatures [Fra89, DDFY94, GRJK07, Sho00]
- Schnorr signatures [SS01]
- (EC)DSA signatures [GJKR01, GGN16]
- BLS signatures [BLS04, Bol03]
- Cramer-Shoup encryption [CG99]
- Many more [SG02, DK05, BBH06, ...]

Not much progress on lattice-based schemes.

Main Results

- Construct Threshold Fully Homomorphic Encryption (TFHE).

Main Results

- Construct Threshold Fully Homomorphic Encryption (TFHE).
- Define a new primitive Universal Thresholdizer (UT).

Main Results

- Construct Threshold Fully Homomorphic Encryption (TFHE).
- Define a new primitive Universal Thresholdizer (UT).
- Show how to use UT as a general tool for constructing threshold cryptosystems.

Main Results

- Construct Threshold Fully Homomorphic Encryption (TFHE).
- Define a new primitive Universal Thresholdizer (UT).
- Show how to use UT as a general tool for constructing threshold cryptosystems.
- New constructions for threshold signatures, threshold PKE, distributed PRFs, functional encryption with distributed key generation, . . .

Defining Threshold Fully Homomorphic Encryption (TFHE)

(Standard) Fully Homomorphic Encryption

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$
- $\mathsf{Encrypt}(\mathsf{pk}, \mu) \to \mathsf{ct}$
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_k) \to \hat{\mathsf{ct}}$
- $\mathsf{Decrypt}(\mathsf{sk}, \hat{\mathsf{ct}}) \to \hat{\mu}$

Defining Threshold Fully Homomorphic Encryption (TFHE)

Threshold Fully Homomorphic Encryption

- $\mathsf{Setup}(1^\lambda, t, N) \rightarrow (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N)$
- $\mathsf{Encrypt}(\mathsf{pk}, \mu) \rightarrow \mathsf{ct}$
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_k) \rightarrow \hat{\mathsf{ct}}$
- $\mathsf{PartDec}(\mathsf{sk}_i, \hat{\mathsf{ct}}) \rightarrow \mathsf{p}_i$
- $\mathsf{FinDec}(\mathsf{pk}, \{\mathsf{p}_i\}) \rightarrow \hat{\mu}$

Defining Threshold Fully Homomorphic Encryption (TFHE)

Threshold Fully Homomorphic Encryption

- $\mathsf{Setup}(1^\lambda, t, N) \to (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N)$
- $\mathsf{Encrypt}(\mathsf{pk}, \mu) \to \mathsf{ct}$
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_k) \to \hat{\mathsf{ct}}$
- $\mathsf{PartDec}(\mathsf{sk}_i, \hat{\mathsf{ct}}) \to \mathsf{p}_i$
- $\mathsf{FinDec}(\mathsf{pk}, \{\mathsf{p}_i\}) \to \hat{\mu}$

Correctness

For any $C : \{0,1\}^k \to \{0,1\}$, $\hat{\mathsf{ct}} \leftarrow \mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$, $|S| \geq t$,

$$\mathsf{FinDec}(\mathsf{pk}, \{\mathsf{PartDec}(\mathsf{sk}_i, \hat{\mathsf{ct}})\}_{i \in S}) = C(\mu_1, \ldots, \mu_k).$$

Defining Threshold Fully Homomorphic Encryption (TFHE)

Threshold Fully Homomorphic Encryption

- $\mathsf{Setup}(1^\lambda, t, N) \to (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N)$
- $\mathsf{Encrypt}(\mathsf{pk}, \mu) \to \mathsf{ct}$
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_k) \to \hat{\mathsf{ct}}$
- $\mathsf{PartDec}(\mathsf{sk}_i, \hat{\mathsf{ct}}) \to \mathsf{p}_i$
- $\mathsf{FinDec}(\mathsf{pk}, \{\mathsf{p}_i\}) \to \hat{\mu}$

Compactness

$$|\mathsf{pk}|, |\mathsf{ct}| \leq \mathsf{poly}(\lambda, d) \quad |\mathsf{p}_i| \leq \mathsf{poly}(\lambda, d, N).$$

Defining Threshold Fully Homomorphic Encryption (TFHE)

Threshold Fully Homomorphic Encryption

- $\mathsf{Setup}(1^\lambda, t, N) \to (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N)$
- $\mathsf{Encrypt}(\mathsf{pk}, \mu) \to \mathsf{ct}$
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_k) \to \hat{\mathsf{ct}}$
- $\mathsf{PartDec}(\mathsf{sk}_i, \hat{\mathsf{ct}}) \to \mathsf{p}_i$
- $\mathsf{FinDec}(\mathsf{pk}, \{\mathsf{p}_i\}) \to \hat{\mu}$

Semantic Security

Standard PKE semantic security
(Adversary given $\{\mathsf{sk}_i\}_{i \in S^*}$ for $|S^*| < t$)

Simulation Security (Real World)

Challenger		Adversary
pk $\mathsf{sk}_1, \ldots, \mathsf{sk}_N$	$\xrightarrow{\mathsf{pk}}$	
	$\xleftarrow{S^*}$	$\lvert S^* \rvert = t - 1$
	$\xrightarrow{\{\mathsf{sk}_i\}_{i \in S^*}}$	
	$\xleftarrow{\mu_1, \ldots, \mu_k}$	
	$\xrightarrow{\mathsf{ct}_1, \ldots, \mathsf{ct}_k}$	
$\hat{\mathsf{ct}} \leftarrow \mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_k)$	$\xleftarrow{(i \in [N], C)}$	
$\mathsf{p}_i \leftarrow \mathsf{PartDec}(\mathsf{sk}_i, \mathsf{ct})$	$\xrightarrow{\mathsf{p}_i}$	

Simulation Security (Ideal World)

Challenger		**Adversary**
pk	$\xrightarrow{\mathsf{pk}}$	
$\mathsf{sk}_1, \ldots, \mathsf{sk}_N$	$\xleftarrow{S^*}$	$\lvert S^* \rvert = t - 1$
	$\xrightarrow{\{\mathsf{sk}_i\}_{i \in S^*}}$	
	$\xleftarrow{\mu_1, \ldots, \mu_k}$	
	$\xrightarrow{\mathsf{ct}_1, \ldots, \mathsf{ct}_k}$	
	$\xleftarrow{(i \in [N], C)}$	
$\mathcal{S}(\{\mathsf{sk}_i\}_{S^*}, \{\mathsf{ct}_i\}_{[k]}, C, C(\mu_1, \ldots, \mu_k))$	$\xrightarrow{\mathsf{p}_i}$	

FHE from Approximate Eigenvector [GSW13]

- Ciphertext $\mathbf{C}$ is a matrix in $\{0,1\}^{m\times m}$
- Secret key $\vec{\mathbf{s}}$ is a vector in $\mathbb{Z}_q^m$

$$\vec{\mathbf{s}} = \left(s_1, \ldots, s_{m-1}, \frac{q}{2}\right)$$

Approximate eigenvector property

$$\mathbf{C} \cdot \vec{\mathbf{s}} = \mu \cdot \vec{\mathbf{s}} + \text{noise}$$

- **Homomorphic addition**: $\mathbf{C}_1 + \mathbf{C}_2$

  $\vec{\mathbf{s}}$ is an eigenvector for $(\mathbf{C}_1 + \mathbf{C}_2)$.
- **Homomorphic multiplicaton**: $\mathbf{C}_1 \cdot \mathbf{C}_2$

  $\vec{\mathbf{s}}$ is an eigenvector for $(\mathbf{C}_1 \cdot \mathbf{C}_2)$

IDEA

Observation: Decryption operation is *linear*.

IDEA

Observation: Decryption operation is *linear*.

Question: Can we use Shamir secret sharing to break up the key $\vec{\mathbf{s}}$?

$$\vec{\mathbf{s}} \to \vec{\mathbf{s}}_1, \ldots, \vec{\mathbf{s}}_N.$$

IDEA

Observation: Decryption operation is *linear*.

Question: Can we use Shamir secret sharing to break up the key $\vec{\mathbf{s}}$?

$$\vec{\mathbf{s}} \to \vec{\mathbf{s}}_1, \ldots, \vec{\mathbf{s}}_N.$$

For any $|S| = t$, $i \in S$, there exists Lagrange coefficient λ_i such that

$$\vec{\mathbf{s}} = \sum_{i \in S} \lambda_i \cdot \vec{\mathbf{s}}_i.$$

IDEA

Define partial decryption

$$\mathrm{PartDec}(\mathsf{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \mathbf{C} \cdot \vec{\mathbf{s}}_i.$$

IDEA

Define partial decryption

$$\mathsf{PartDec}(\mathsf{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \mathbf{C} \cdot \vec{\mathbf{s}}_i.$$

Define final decryption

$$\mathsf{FinDec}(\mathsf{pk}, \{\mathbf{C} \cdot \vec{\mathbf{s}}_i\}_S) = \sum_{i \in S} \lambda_i \cdot (\mathbf{C} \cdot \vec{\mathbf{s}}_i)$$

IDEA

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \mathbf{C} \cdot \vec{\mathbf{s}}_i.$$

Define final decryption

$$\begin{aligned}\mathrm{FinDec}(\mathrm{pk}, \{\mathbf{C} \cdot \vec{\mathbf{s}}_i\}_S) &= \sum_{i \in S} \lambda_i \cdot (\mathbf{C} \cdot \vec{\mathbf{s}}_i) \\ &= \mathbf{C} \cdot \sum_{i \in S} \lambda_i \cdot \vec{\mathbf{s}}_i\end{aligned}$$

IDEA

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \mathbf{C} \cdot \vec{\mathbf{s}}_i.$$

Define final decryption

$$\begin{aligned}\mathrm{FinDec}(\mathrm{pk}, \{\mathbf{C} \cdot \vec{\mathbf{s}}_i\}_S) &= \sum_{i \in S} \lambda_i \cdot (\mathbf{C} \cdot \vec{\mathbf{s}}_i) \\ &= \mathbf{C} \cdot \sum_{i \in S} \lambda_i \cdot \vec{\mathbf{s}}_i\end{aligned}$$

IDEA

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \mathbf{C} \cdot \vec{\mathbf{s}}_i.$$

Define final decryption

$$\begin{aligned}\mathrm{FinDec}(\mathrm{pk}, \{\mathbf{C} \cdot \vec{\mathbf{s}}_i\}_S) &= \sum_{i \in S} \lambda_i \cdot (\mathbf{C} \cdot \vec{\mathbf{s}}_i) \\ &= \mathbf{C} \cdot \sum_{i \in S} \lambda_i \cdot \vec{\mathbf{s}}_i \\ &= \mathbf{C} \cdot \vec{\mathbf{s}}\end{aligned}$$

IDEA

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \mathbf{C} \cdot \vec{\mathbf{s}}_i.$$

Define final decryption

$$\begin{aligned}
\mathrm{FinDec}(\mathrm{pk}, \{\mathbf{C} \cdot \vec{\mathbf{s}}_i\}_S) &= \sum_{i \in S} \lambda_i \cdot (\mathbf{C} \cdot \vec{\mathbf{s}}_i) \\
&= \mathbf{C} \cdot \sum_{i \in S} \lambda_i \cdot \vec{\mathbf{s}}_i \\
&= \mathbf{C} \cdot \vec{\mathbf{s}} \\
&= \mu \cdot \vec{\mathbf{s}} + \mathrm{noise}
\end{aligned}$$

IDEA

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \mathbf{C} \cdot \vec{\mathbf{s}}_i.$$

Define final decryption

$$\begin{aligned}\mathrm{FinDec}(\mathrm{pk}, \{\mathbf{C} \cdot \vec{\mathbf{s}}_i\}_S) &= \sum_{i \in S} \lambda_i \cdot (\mathbf{C} \cdot \vec{\mathbf{s}}_i) \\ &= \mathbf{C} \cdot \sum_{i \in S} \lambda_i \cdot \vec{\mathbf{s}}_i \\ &= \mathbf{C} \cdot \vec{\mathbf{s}} \\ &= \mu \cdot \vec{\mathbf{s}} + \mathrm{noise}\end{aligned}$$

Slight Problem: Decryption leaks $\vec{\mathbf{s}}$!

IDEA

Let

$$\mathbf{C} = \begin{bmatrix} \vdots & & \vdots \\ \mathbf{c}_1 & \cdots & \mathbf{c}_m \\ \vdots & & \vdots \end{bmatrix}.$$

Define partial decryption

$$\mathrm{PartDec}(\mathsf{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle .$$

Define final decryption

$$\begin{aligned} \mathrm{FinDec}(\mathsf{pk}, \{\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle\}_S) &= \sum_{i \in S} \lambda_i \cdot \langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle \\ &= \left\langle \mathbf{c}_m, \sum_{i \in S} \lambda_i \cdot \vec{\mathbf{s}}_i \right\rangle \\ &= \langle \mathbf{c}_m, \vec{\mathbf{s}} \rangle \\ &= \frac{q}{2} \cdot \mu + \text{noise} \end{aligned}$$

IDEA

Problem: $\mathbf{c}_m$ is a *public vector*!

Every partial decryption $\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle$ leaks information about $\vec{\mathbf{s}}_i$.

IDEA

Problem: $\mathbf{c}_m$ is a *public vector*!

Every partial decryption $\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle$ leaks information about $\vec{\mathbf{s}}_i$.

Solution(?): Can we add some additional noise?

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{c}_m, \vec{\mathbf{s}}_i) = \langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle \, .$$

IDEA

Problem: $\mathbf{c}_m$ is a *public vector*!

Every partial decryption $\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle$ leaks information about $\vec{\mathbf{s}}_i$.

Solution(?): Can we add some additional noise?

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{c}_m, \vec{\mathbf{s}}_i) = \langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \text{noise}.$$

IDEA

Problem: $\mathbf{c}_m$ is a *public vector*!

Every partial decryption $\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle$ leaks information about $\vec{\mathbf{s}}_i$.

Solution(?): Can we add some additional noise?

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{c}_m, \vec{\mathbf{s}}_i) = \langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \textcolor{red}{\mathrm{noise}}.$$

Still Problem: Final decryption

$$\mathrm{FinDec}(\mathrm{pk}, \{\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \mathrm{noise}\}_S) = \sum_{i \in S} \lambda_i \cdot (\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \textcolor{red}{\mathrm{noise}_i})$$

IDEA

Problem: $\mathbf{c}_m$ is a *public vector*!

Every partial decryption $\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle$ leaks information about $\vec{\mathbf{s}}_i$.

Solution(?): Can we add some additional noise?

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{c}_m, \vec{\mathbf{s}}_i) = \langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \mathrm{noise}.$$

Still Problem: Final decryption

$$\begin{aligned}
\mathrm{FinDec}(\mathrm{pk}, \{\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \mathrm{noise}\}_S) &= \sum_{i \in S} \lambda_i \cdot (\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \mathrm{noise}_i) \\
&= \sum_{i \in S} \lambda_i \cdot \langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \sum_{i \in S} \lambda_i \cdot \mathrm{noise}_i \\
&= \langle \mathbf{c}_m, \vec{\mathbf{s}} \rangle + \mathrm{BIG} \\
&= \frac{q}{2} \cdot \mu \cdot \vec{\mathbf{s}} + \mathrm{BIG}
\end{aligned}$$

Problem of Secret Sharing

Two methods of overcoming *noise blow-up*:

1. Define a linear secret sharing scheme with low-norm reconstruction coefficients

Problem of Secret Sharing

Two methods of overcoming *noise blow-up*:

1. Define a linear secret sharing scheme with low-norm reconstruction coefficients
2. Change the scheme itself using clearing out denominators trick

$\{0,1\}$-LSSS

Linear secret sharing scheme for $\mathsf{k} \in \mathbb{Z}_q$

- $\mathsf{Share}(\mathsf{k}, \phi) \to (\mathsf{s}_1, \ldots, \mathsf{s}_N) \in \mathbb{Z}_q^N$
- $\mathsf{Combine}(\{\mathsf{s}_i\}_S)$:
  - For any set $\phi(S) = 1$, there exists efficiently computable coefficients $c_i \in \mathbb{Z}_q$ such that,

$$\mathsf{k} = \sum_{i \in S} c_i \cdot \mathsf{s}_i.$$

$\{0,1\}$-LSSS

Linear secret sharing scheme for $\mathsf{k} \in \mathbb{Z}_q$

- $\mathsf{Share}(\mathsf{k}, \phi) \to (\mathsf{s}_1, \ldots, \mathsf{s}_N) \in \mathbb{Z}_q^N$
- $\mathsf{Combine}(\{\mathsf{s}_i\}_S)$:
  - For any set $\phi(S) = 1$, there exists efficiently computable coefficients $c_i \in \mathbb{Z}_q$ such that,

$$\mathsf{k} = \sum_{i \in S} c_i \cdot \mathsf{s}_i.$$

Example: Shamir secret sharing scheme

$$\phi_t(S) = 1 \Leftrightarrow |S| \geq t.$$

$\{0, 1\}$-LSSS

Linear secret sharing scheme for $\mathsf{k} \in \mathbb{Z}_q$

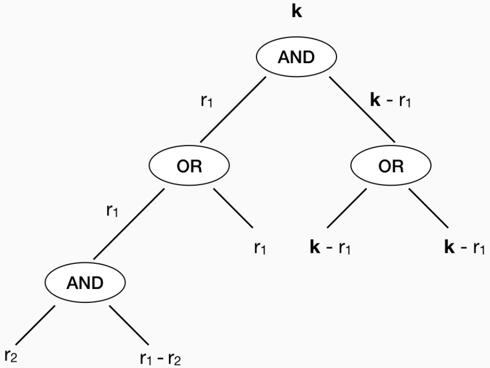
- $\mathsf{Share}(\mathsf{k}, \phi) \rightarrow (\mathsf{s}_1, \ldots, \mathsf{s}_N) \in \mathbb{Z}_q^N$
- $\mathsf{Combine}(\{\mathsf{s}_i\}_S)$:
  - For any set $\phi(S) = 1$, there exists efficiently computable coefficients $c_i \in \mathbb{Z}_q$ such that,

$$\mathsf{k} = \sum_{i \in S} c_i \cdot \mathsf{s}_i .$$

Define $\{0, 1\}$-LSSS as a class of linear secret sharing schemes where the *reconstruction coefficients* are always binary.

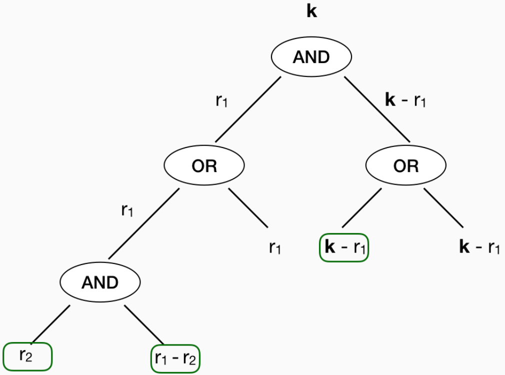
Question: How expressive is $\{0, 1\}$-LSSS?

Monotone Boolean Formulas



$$s_1 = r_2 \qquad s_4 = \mathbf{k} - r_1$$
$$s_2 = r_1 - r_2 \qquad s_5 = \mathbf{k} - r_1$$
$$s_3 = r_1$$

Monotone Boolean Formulas



$$s_1 = r_2 \qquad s_4 = \mathbf{k} - r_1$$
$$s_2 = r_1 - r_2 \qquad s_5 = \mathbf{k} - r_1$$
$$s_3 = r_1$$

$\{0,1\}$-LSSS

- $\{0,1\}$-LSSS contains access structures induced by monotone Boolean formulas.

$\{0,1\}$-LSSS

- $\{0,1\}$-LSSS contains access structures induced by monotone Boolean formulas.
- [Val84, Gol14] show that threshold function expressible by monotone Boolean formulas.

Recap

Use $\{0,1\}$-LSSS to break up FHE key $\vec{\mathbf{s}}$

$$\vec{\mathbf{s}} \rightarrow \vec{\mathbf{s}}_1, \ldots, \vec{\mathbf{s}}_N$$

Recap

Use $\{0,1\}$-LSSS to break up FHE key $\vec{\mathbf{s}}$

$$\vec{\mathbf{s}} \to \vec{\mathbf{s}}_1, \ldots, \vec{\mathbf{s}}_N$$

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \text{noise}.$$

Recap

Use $\{0,1\}$-LSSS to break up FHE key $\vec{\mathbf{s}}$

$$\vec{\mathbf{s}} \to \vec{\mathbf{s}}_1, \ldots, \vec{\mathbf{s}}_N$$

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \text{noise}.$$

Define final decryption

$$\begin{aligned} \mathrm{FinDec}(\mathrm{pk}, \{\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \text{noise}_i\}_S) &= \sum_{i \in S} c_i \cdot (\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \text{noise}_i) \\ &= \langle \mathbf{c}_m, \vec{\mathbf{s}} \rangle + \underbrace{\text{noise} + \sum_{i \in S} c_i \cdot \text{noise}_i}_{\text{small}} \end{aligned}$$

Recap

Use $\{0,1\}$-LSSS to break up FHE key $\vec{\mathbf{s}}$

$$\vec{\mathbf{s}} \to \vec{\mathbf{s}}_1, \ldots, \vec{\mathbf{s}}_N$$

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \text{noise}.$$

Define final decryption

$$\begin{aligned}\mathrm{FinDec}(\mathrm{pk}, \{\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \text{noise}_i\}_S) &= \sum_{i \in S} c_i \cdot (\langle \mathbf{c}_m, \vec{\mathbf{s}}_i \rangle + \text{noise}_i) \\ &= \langle \mathbf{c}_m, \vec{\mathbf{s}} \rangle + \underbrace{\text{noise} + \sum_{i \in S} c_i \cdot \text{noise}_i}_{\text{small}}\end{aligned}$$

Note: Requires careful security analysis!

Clearing out Denominators

Expressing threshold circuit in monotone Boolean formula expensive!

Circuit size $O(N^{5.2}) \Rightarrow$ partial key $|\mathsf{sk}_i| \leq O(N^{4.2})$ on average.

Question: Can we do better?

Idea: Use the technique of clearing out the denominators [Sho00,ABVVW12]

Lemma

For any Lagrange coefficients λ_i,

$$|(N!) \cdot \lambda_i| \leq (N!)^3.$$

Clearing out Denominators

Use Shamir secret sharing to break up FHE key $\vec{\mathbf{s}}$

$$\vec{\mathbf{s}} \to \vec{\mathbf{s}}_1, \ldots, \vec{\mathbf{s}}_N$$

Define partial decryption

$$\mathrm{PartDec}(\mathrm{pk}, \mathbf{C}, \vec{\mathbf{s}}_i) = \mathbf{C} \cdot \vec{\mathbf{s}}_i + (N!)^2 \cdot \mathrm{noise}.$$

Define final decryption

$$\begin{aligned} \mathrm{FinDec}(\mathrm{pk}, \{\mathbf{C} \cdot \vec{\mathbf{s}}_i\}_S) &= \sum_{i \in S} \lambda_i \cdot (\mathbf{C} \cdot \vec{\mathbf{s}}_i + (N!)^2 \cdot \mathrm{noise}_i) \\ &= C \cdot \vec{\mathbf{s}} + \underbrace{\mathrm{noise} + \sum_{i \in S} \lambda_i \cdot (N!)^2 \cdot \mathrm{noise}_i}_{\textit{relatively}\ \text{small}} \end{aligned}$$

Universal Thresholdizer

Universal Thresholdizer

- $\mathsf{Setup}(1^\lambda, t, N, x) \to (\mathsf{pp}, \mathsf{s}_1, \ldots, \mathsf{s}_N)$ for $x \in \{0,1\}^k$
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{s}_i, C) \to \mathsf{p}_i$
- $\mathsf{Combine}(\mathsf{pp}, \{\mathsf{p}_i\}) \to C(x)$

Universal Thresholdizer

Universal Thresholdizer

- $\mathsf{Setup}(1^\lambda, t, N, x) \to (\mathsf{pp}, \mathsf{s}_1, \dots, \mathsf{s}_N)$ for $x \in \{0,1\}^k$
  - $\mathsf{Setup}(1^\lambda, t, N) \to (\mathsf{pk}, \mathsf{sk}_1, \dots, \mathsf{sk}_N)$
  - $\mathsf{Encrypt}(\mathsf{pk}, x) \to \mathsf{ct}$

$$\mathsf{pp} = (\mathsf{pk}, \mathsf{ct}) \qquad \mathsf{s}_i = \mathsf{sk}_i.$$

- $\mathsf{Eval}(\mathsf{pp}, \mathsf{s}_i, C) \to \mathsf{p}_i$
- $\mathsf{Combine}(\mathsf{pp}, \{\mathsf{p}_i\}) \to C(x)$

Universal Thresholdizer

Universal Thresholdizer

- $\mathsf{Setup}(1^\lambda, t, N, x) \to (\mathsf{pp}, \mathsf{s}_1, \dots, \mathsf{s}_N)$ for $x \in \{0,1\}^k$
  - $\mathsf{Setup}(1^\lambda, t, N) \to (\mathsf{pk}, \mathsf{sk}_1, \dots, \mathsf{sk}_N)$
  - $\mathsf{Encrypt}(\mathsf{pk}, x) \to \mathsf{ct}$

$$\mathsf{pp} = (\mathsf{pk}, \mathsf{ct}) \qquad \mathsf{s}_i = \mathsf{sk}_i.$$

- $\mathsf{Eval}(\mathsf{pp}, \mathsf{s}_i, C) \to \mathsf{p}_i$
  - $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}) \to \hat{\mathsf{ct}}$
  - $\mathsf{PartDec}(\mathsf{sk}_i, \hat{\mathsf{ct}}) \to \mathsf{p}_i$
- $\mathsf{Combine}(\mathsf{pp}, \{\mathsf{p}_i\}) \to C(x)$

Universal Thresholdizer

Universal Thresholdizer

- $\mathsf{Setup}(1^\lambda, t, N, x) \to (\mathsf{pp}, \mathsf{s}_1, \ldots, \mathsf{s}_N) \qquad \text{for } x \in \{0,1\}^k$
  - $\mathsf{Setup}(1^\lambda, t, N) \to (\mathsf{pk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N)$
  - $\mathsf{Encrypt}(\mathsf{pk}, x) \to \mathsf{ct}$

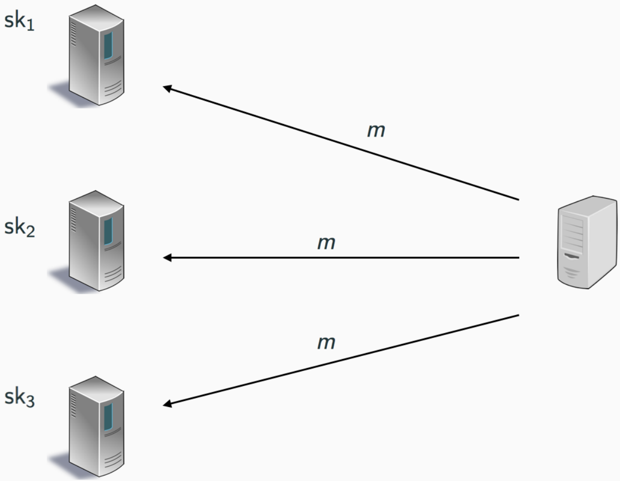
$$\mathsf{pp} = (\mathsf{pk}, \mathsf{ct}) \qquad \mathsf{s}_i = \mathsf{sk}_i.$$

- $\mathsf{Eval}(\mathsf{pp}, \mathsf{s}_i, C) \to \mathsf{p}_i$
  - $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}) \to \hat{\mathsf{ct}}$
  - $\mathsf{PartDec}(\mathsf{sk}_i, \hat{\mathsf{ct}}) \to \mathsf{p}_i$
- $\mathsf{Combine}(\mathsf{pp}, \{\mathsf{p}_i\}) \to C(x)$
  - $\mathsf{FinDec}(\mathsf{pk}, \{\mathsf{p}_i\}) \to C(x)$

Threshold Signatures

$\mathsf{pp} = (\mathsf{pk}, \mathsf{Enc}(\mathsf{sk_{sign}}))$

$C_m(\mathsf{sk}) = \mathsf{Sign}(\mathsf{sk}, m)$



Threshold Signatures

$\mathsf{pp} = (\mathsf{pk}, \mathsf{Enc}(\mathsf{sk}_{\mathsf{sign}}))$

$C_m(\mathsf{sk}) = \mathsf{Sign}(\mathsf{sk}, m)$

sk_1

$\mathsf{Enc}(\sigma)$

sk_2

$\mathsf{Enc}(\sigma)$

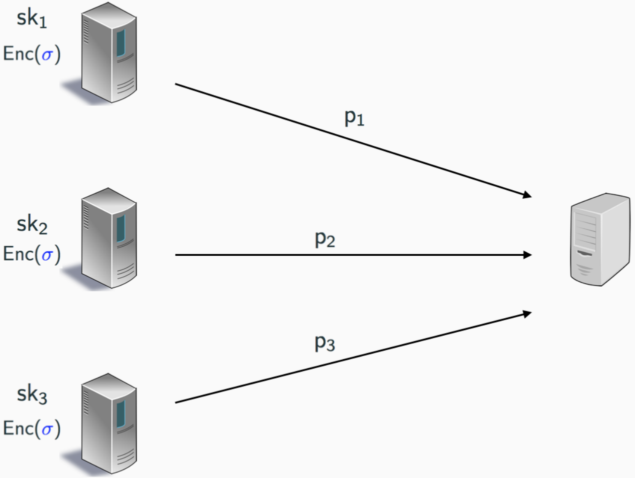
sk_3

$\mathsf{Enc}(\sigma)$

Threshold Signatures

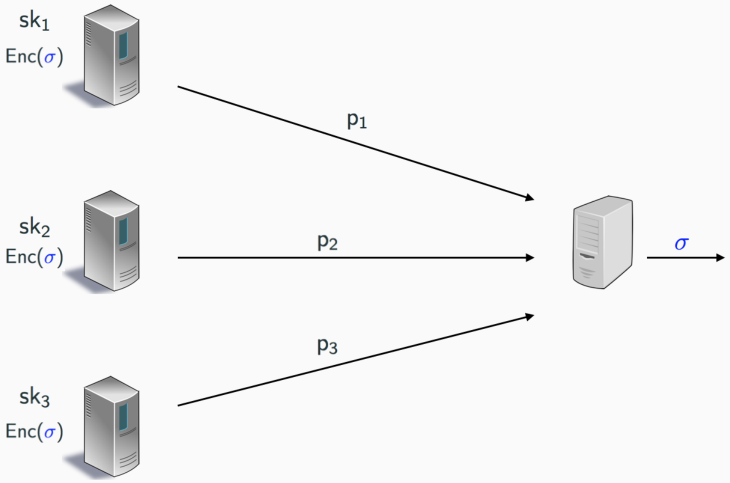
$\mathsf{pp} = (\mathsf{pk}, \mathsf{Enc}(\mathsf{sk}_{\mathsf{sign}}))$

$C_m(\mathsf{sk}) = \mathsf{Sign}(\mathsf{sk}, m)$



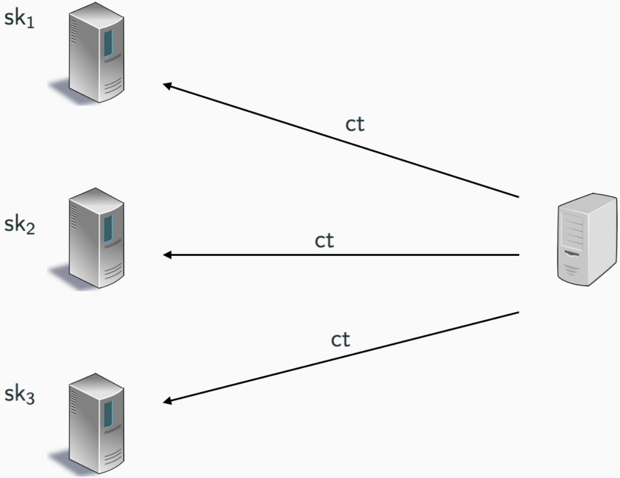
Threshold Signatures

$\mathsf{pp} = (\mathsf{pk}, \mathsf{Enc}(\mathsf{sk}_{\mathsf{sign}}))$ $\quad$ $C_m(\mathsf{sk}) = \mathsf{Sign}(\mathsf{sk}, m)$



Threshold PKE

$\mathsf{pp} = (\mathsf{pk}, \mathsf{Enc}(\mathsf{sk}_{\mathsf{Dec}}))$



Threshold PKE

$\mathsf{pp} = (\mathsf{pk}, \mathsf{Enc}(\mathsf{sk}_{\mathsf{Dec}}))$

$C_{\mathsf{ct}}(\mathsf{sk}) = \mathsf{Dec}(\mathsf{sk}, m)$

sk_1

$\mathsf{Enc}(\mu)$

sk_2

$\mathsf{Enc}(\mu)$

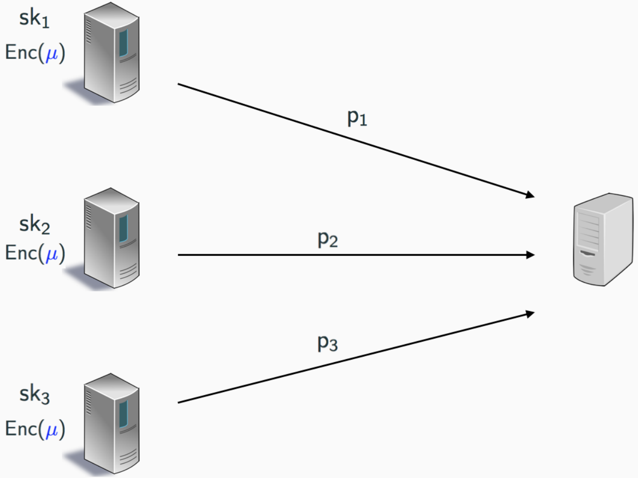
sk_3

$\mathsf{Enc}(\mu)$

Threshold PKE

$\mathsf{pp} = (\mathsf{pk}, \mathsf{Enc}(\mathsf{sk}_{\mathsf{Dec}}))$

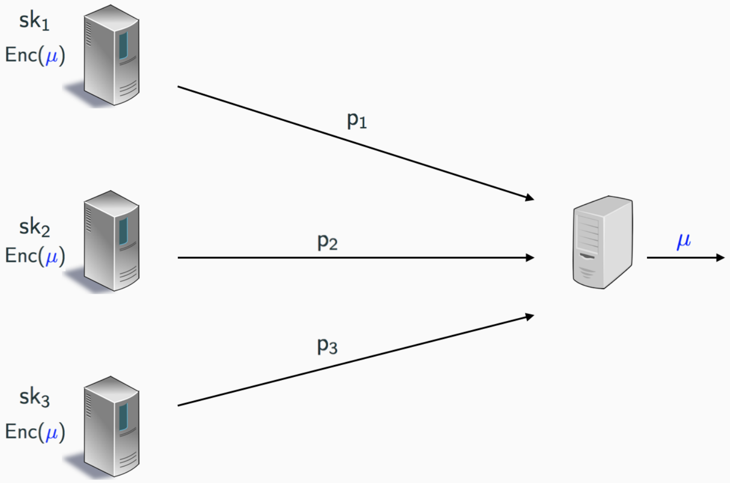
$C_{\mathsf{ct}}(\mathsf{sk}) = \mathsf{Dec}(\mathsf{sk}, m)$



Threshold PKE

$\mathsf{pp} = (\mathsf{pk}, \mathsf{Enc}(\mathsf{sk}_{\mathsf{Dec}}))$

$C_{\mathsf{ct}}(\mathsf{sk}) = \mathsf{Dec}(\mathsf{sk}, m)$



To Conclude...

Did not cover:

- Technical challenges in the analysis
- Decentralizing threshold FHE
- More applications like function secret sharing, distributed PRFs, ...

To Conclude...

Did not cover:

- Technical challenges in the analysis
- Decentralizing threshold FHE
- More applications like function secret sharing, distributed PRFs, ...

Open Problems:

- Decentralizing universal thresholdizer?
- Direct constructions of threshold signatures or threshold PKE?

To Conclude...

Did not cover:

- Technical challenges in the analysis
- Decentralizing threshold FHE
- More applications like function secret sharing, distributed PRFs, ...

Open Problems:

- Decentralizing universal thresholdizer?
- Direct constructions of threshold signatures or threshold PKE?

Thanks!