NTNU | Norwegian University of Science and Technology

QUANTUM-SAFE THRESHOLD SIGNATURES

Kamil Doruk Gur, Patrick Hough, Jonathan Katz,
Caroline Sandsbråten and **Tjerand Silde**

Contents

Contents

Threshold Cryptography

The goal is that secrets are shared between n parties, and that any threshold $1 \leq t \leq n$ can jointly compute a decryption or signature based on their shares.

This gives security against an adversary corrupting at most $t - 1$ parties which cannot complete the computation on its own, and robustness if at least t honest parties are available for the computation to be completed.

Applications

- sign transactions and legal documents
- sign authentication challenges or certificates
- decrypt ballots in an electronic voting system
- run pre-processing phases for MPC protocols

Contents

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \mid \mathbf{I}] \in R_q^{k\times(\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \mid \mathbf{I}] \in R_q^{k\times(\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

1. The prover samples a short vector $\boldsymbol{r} \in R_q^{\ell+k}$ and sends $\mathbf{w} := \bar{\mathbf{A}}\boldsymbol{r}$.

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \mid \mathbf{I}] \in R_q^{k\times(\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

1. The prover samples a short vector $\boldsymbol{r} \in R_q^{\ell+k}$ and sends $\mathbf{w} := \bar{\mathbf{A}}\boldsymbol{r}$.
2. The verifier responds with a short challenge $c \in \mathcal{C} \subset R_q$.

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \mid \mathbf{I}] \in R_q^{k\times(\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

1. The prover samples a short vector $\boldsymbol{r} \in R_q^{\ell+k}$ and sends $\boldsymbol{w} := \bar{\mathbf{A}}\boldsymbol{r}$.
2. The verifier responds with a short challenge $c \in \mathcal{C} \subset R_q$.
3. The prover responds with a short vector $\boldsymbol{z} := c \cdot \mathbf{s} + \boldsymbol{r}$.

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \mid \mathbf{I}] \in R_q^{k\times(\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

1. The prover samples a short vector $\boldsymbol{r} \in R_q^{\ell+k}$ and sends $\mathbf{w} := \bar{\mathbf{A}}\boldsymbol{r}$.
2. The verifier responds with a short challenge $c \in \mathcal{C} \subset R_q$.
3. The prover responds with a short vector $\boldsymbol{z} := c \cdot \mathbf{s} + \boldsymbol{r}$.
4. $\rightarrow$The prover might abort because of rejection sampling.

Basic Signature Scheme

The private key is a short $\mathbf{s} \in R_q^{\ell+k}$, and the verification key consists of a matrix $\bar{\mathbf{A}} := [\mathbf{A} \,|\, \mathbf{I}] \in R_q^{k \times (\ell+k)}$ and vector $\boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s}$. The protocol proceeds as follows:

1. The prover samples a short vector $\boldsymbol{r} \in R_q^{\ell+k}$ and sends $\mathbf{w} := \bar{\mathbf{A}}\boldsymbol{r}$.
2. The verifier responds with a short challenge $c \in \mathcal{C} \subset R_q$.
3. The prover responds with a short vector $\boldsymbol{z} := c \cdot \mathbf{s} + \boldsymbol{r}$.
4. $\rightarrow$The prover might abort because of rejection sampling.
5. The verifier accepts iff $\boldsymbol{z}$ is short and $\bar{\mathbf{A}}\boldsymbol{z} = c \cdot \boldsymbol{y} + \mathbf{w}$.
6. $\rightarrow$ Non-interactive signature scheme if $c = H(\mathsf{pk}, \mathbf{w}, m)$.

Basic n-out-of-n Threshold Scheme

The ith signer holds short vector $\mathbf{s}_i$ where $\mathbf{s} = \sum_{i \in [n]} \mathbf{s}_i$ is the private key. Then, the n signers can run a distributed, two-round signing protocol as follows:

Basic n-out-of-n Threshold Scheme

The ith signer holds short vector $\mathbf{s}_i$ where $\mathbf{s} = \sum_{i \in [n]} \mathbf{s}_i$ is the private key. Then, the n signers can run a distributed, two-round signing protocol as follows:

1. The ith signer chooses a short vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$.

Basic n-out-of-n Threshold Scheme

The ith signer holds short vector $\mathbf{s}_i$ where $\mathbf{s} = \sum_{i\in[n]} \mathbf{s}_i$ is the private key. Then, the n signers can run a distributed, two-round signing protocol as follows:

1. The ith signer chooses a short vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$.
2. Each signer computes $\boldsymbol{w} := \sum_{i\in[n]} \boldsymbol{w}_i$ followed by $c := H(\boldsymbol{w})$. The ith signer then sends $\boldsymbol{z}_i := c \cdot \mathbf{s}_i + \boldsymbol{r}_i$.

Basic n-out-of-n Threshold Scheme

The ith signer holds short vector $\mathbf{s}_i$ where $\mathbf{s} = \sum_{i \in [n]} \mathbf{s}_i$ is the private key. Then, the n signers can run a distributed, two-round signing protocol as follows:

1. The ith signer chooses a short vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$.
2. Each signer computes $\boldsymbol{w} := \sum_{i \in [n]} \boldsymbol{w}_i$ followed by $c := H(\boldsymbol{w})$. The ith signer then sends $\boldsymbol{z}_i := c \cdot \mathbf{s}_i + \boldsymbol{r}_i$.
3. Each signer then computes $\boldsymbol{z} := \sum_{i \in [n]} \boldsymbol{z}_i$ and outputs the signature $(c, \boldsymbol{z})$.

Issues with Secret Sharing

Issues with Secret Sharing

- The shared secret must be short for SIS to be hard

Issues with Secret Sharing

- The shared secret must be short for SIS to be hard
- Individual secrets must be short to allow rejection sampling

Issues with Secret Sharing

- The shared secret must be short for SIS to be hard
- Individual secrets must be short to allow rejection sampling
- The sum of short elements is also short, but...

Issues with Secret Sharing

- The shared secret must be short for SIS to be hard
- Individual secrets must be short to allow rejection sampling
- The sum of short elements is also short, but...
- Shamir secret shared elements are uniformly random

Issues with Random Oracles

Issues with Random Oracles

- Fiat-Shamir signatures require a random oracle to produce challenges, and we cannot evaluate th RO using MPC, ZKP, or FHE in a black-box way.

Issues with Random Oracles

- Fiat-Shamir signatures require a random oracle to produce challenges, and we cannot evaluate th RO using MPC, ZKP, or FHE in a black-box way.

- There are schemes computing threshold signatures using generic FHE (heavy computation and evaluates the RO circuit) or MPC (many rounds of commutation and distributed rejection sampling), but these are not ideal.

Issues with Random Oracles

- Fiat-Shamir signatures require a random oracle to produce challenges, and we cannot evaluate th RO using MPC, ZKP, or FHE in a black-box way.
- There are schemes computing threshold signatures using generic FHE (heavy computation and evaluates the RO circuit) or MPC (many rounds of commutation and distributed rejection sampling), but these are not ideal.
- We need a homomorphism to share and combine secrets, but we want to evaluate the random oracle on public input (communicated messages).

Issues with Zero-Knowledge

Issues with Zero-Knowledge

- Signatures are (honest-verifier) zero-knowledge when no parties abort.

Issues with Zero-Knowledge

- Signatures are (honest-verifier) zero-knowledge when no parties abort.
- Then the commit message cannot be sent in the clear if anyone aborts.

Issues with Zero-Knowledge

- Signatures are (honest-verifier) zero-knowledge when no parties abort.
- Then the commit message cannot be sent in the clear if anyone aborts.
- We only learn if anyone aborts after we have computed the challenge...

Contents

BGV Encryption

The BGV encryption scheme consists of the following algorithms:

BGV Encryption

The BGV encryption scheme consists of the following algorithms:

- $\mathsf{KGen}_{\mathsf{BGV}}$: Sample a uniform element $a \in R_q$ along with $s, e \leftarrow D_{\mathsf{KGen}}$, and output the public key $\mathsf{pk} := (a, b) = (a, as + pe)$ and secret key $\mathsf{sk} := s$.

BGV Encryption

The BGV encryption scheme consists of the following algorithms:

- $\mathsf{KGen}_{\mathsf{BGV}}$: Sample a uniform element $a \in R_q$ along with $s, e \leftarrow D_{\mathsf{KGen}}$, and output the public key $\mathsf{pk} := (a, b) = (a, as + pe)$ and secret key $\mathsf{sk} := s$.
- $\mathsf{Enc}_{\mathsf{BGV}}$: On input a public key $\mathsf{pk} = (a, b)$ and a message $m \in R_p$, sample $r, e', e'' \leftarrow D_{\mathsf{Enc}}$ and output the ciphertext $(u, v) = (ar + pe', br + pe'' + m)$.

BGV Encryption

The BGV encryption scheme consists of the following algorithms:

- $\mathsf{KGen}_{\mathsf{BGV}}$: Sample a uniform element $a \in R_q$ along with $s, e \leftarrow D_{\mathsf{KGen}}$, and output the public key $\mathsf{pk} := (a, b) = (a, as + pe)$ and secret key $\mathsf{sk} := s$.
- $\mathsf{Enc}_{\mathsf{BGV}}$: On input a public key $\mathsf{pk} = (a, b)$ and a message $m \in R_p$, sample $r, e', e'' \leftarrow D_{\mathsf{Enc}}$ and output the ciphertext $(u, v) = (ar + pe', br + pe'' + m)$.
- $\mathsf{Dec}_{\mathsf{BGV}}$: On input a secret key $\mathsf{sk} = s$ and a ciphertext (u, v), output the message $m := (v - su \bmod q) \bmod p$.

BGV DistKeyGen

The distributed key generation protocol for BGV works as follows:

BGV DistKeyGen

The distributed key generation protocol for BGV works as follows:

1. $\mathcal{P}_i$ samples s_i and e_i from a distribution D_{KGen}, computes $b_i := as_i + pe_i$.

BGV DistKeyGen

The distributed key generation protocol for BGV works as follows:

1. $\mathcal{P}_i$ samples s_i and e_i from a distribution D_{KGen}, computes $b_i := as_i + pe_i$.
2. $\mathcal{P}_i$ secret shares s_i into $\{s_{i,j}\}_{j\in[n]}$ using t-out-of-n Shamir secret sharing. For each j, $\mathcal{P}_i$ sends $s_{i,j}$ and b_i to party $\mathcal{P}_j$ over a secure channel.

BGV DistKeyGen

The distributed key generation protocol for BGV works as follows:

1. $\mathcal{P}_i$ samples s_i and e_i from a distribution D_{KGen}, computes $b_i := as_i + pe_i$.
2. $\mathcal{P}_i$ secret shares s_i into $\{s_{i,j}\}_{j \in [n]}$ using t-out-of-n Shamir secret sharing. For each j, $\mathcal{P}_i$ sends $s_{i,j}$ and b_i to party $\mathcal{P}_j$ over a secure channel.
3. $\mathcal{P}_i$ computes $b := \sum b_j$, $s'_i = \sum s_{j,i}$, and outputs $\mathsf{pk} = (a, b)$ and $\mathsf{sk}_i = s'_i$.

BGV DistKeyGen

The distributed key generation protocol for BGV works as follows:

1. $\mathcal{P}_i$ samples s_i and e_i from a distribution D_{KGen}, computes $b_i := as_i + pe_i$.
2. $\mathcal{P}_i$ secret shares s_i into $\{s_{i,j}\}_{j\in[n]}$ using t-out-of-n Shamir secret sharing. For each j, $\mathcal{P}_i$ sends $s_{i,j}$ and b_i to party $\mathcal{P}_j$ over a secure channel.
3. $\mathcal{P}_i$ computes $b := \sum b_j$, $s'_i = \sum s_{j,i}$, and outputs $\mathsf{pk} = (a, b)$ and $\mathsf{sk}_i = s'_i$.
4. $\rightarrow$ The protocol can be made actively secure with commitments and ZKPs.

BGV DistKeyGen

$\underline{\mathsf{DKGen}(a_{\mathcal{E}}, t, n)}$

$s_i, e_i \leftarrow \chi_{\mathsf{tern}}, \quad b_i := a_{\mathcal{E}} s_i + p e_i, \quad h_i := H(i, b_i)$

$$\xrightarrow{\quad h_i \quad}$$

$$\xleftarrow{\quad \{h_j\}_{j \neq i} \quad}$$

$\{s_{i,j}\}_{j\in[n]} \leftarrow \mathsf{Share}_{t,n}(s_i), \ \{e_{i,j}\}_{j\in[n]} \leftarrow \mathsf{Share}_{t,n}(e_i)$

$\forall j \in [n] : b_{i,j} := a_{\mathcal{E}} s_{i,j} + p e_{i,j}$

$\forall j \in [n] : \rho_{i,j} \leftarrow S_\rho, \ \mathsf{com}_{i,j} := \mathsf{Com}(s_{i,j}; \rho_{i,j})$

Compute NIZK proof π_i for shortness and linear relations

$$\xrightarrow{\quad b_i, \{b_{i,j}, \mathsf{com}_{i,j}\}_{j\in[n]}, \pi_i, \quad [s_{i,j}, e_{i,j}, \rho_{i,j}]_{j\in[n]} \quad}$$

$$\xleftarrow{\quad \left\{b_j, \{b_{j,k}, \mathsf{com}_{j,k}\}_{k\in[n]}, \pi_j, \quad [s_{j,i}, e_{j,i}, \rho_{j,i}]\right\}_{j\neq i} \quad}$$

if any $h_j \neq H(j, b_j)$: **abort** (j)

if any π_j **is invalid**: **abort** (j)

if any $0 = \mathsf{Open}(\mathsf{com}_{j,i}, s_{j,i}, \rho_{j,i})$: **abort** (j)

$b_{\mathcal{E}} := \sum_j b_j, \quad \mathsf{sk}_i := \sum_j s_{j,i}, \quad \rho_i = \sum_j \rho_{j,i}$

$\forall j \in [n] : \mathsf{com}_j := \sum_{k\in[n]} \mathsf{com}_{j,k}$

return $\mathsf{pk}_{\mathcal{E}} := (a_{\mathcal{E}}, b_{\mathcal{E}}, \{\mathsf{com}_j\}_{j\in[n]}), \quad \mathsf{sk}'_i := (\mathsf{sk}_i, \rho_i)$

Fig. 2. Actively secure key-generation protocol, from the point of view of $\mathcal{P}_i$. The elements in square brackets with subscript j are sent to $\mathcal{P}_j$ over a private channel.

BGV TDec

The threshold decryption procedure for BGV works as follows:

BGV TDec

The threshold decryption procedure for BGV works as follows:

TDec On input a ciphertext $\mathsf{ctx} = (u, v)$, a decryption key share $\mathsf{sk}_i = s_i$, and a set of users $\mathcal{U}$ of size t, compute $m_i := \lambda_i s u$ using Lagrange coefficient λ_i.

Sample noise $E_i \leftarrow R_q$ s.t $\|E_i\|_\infty \leq 2^{\mathsf{sec}} B_{\mathsf{Dec}}$, then output $\mathsf{ds}_i := m_i + pE_i$.

BGV TDec

The threshold decryption procedure for BGV works as follows:

TDec On input a ciphertext $\mathsf{ctx} = (u, v)$, a decryption key share $\mathsf{sk}_i = s_i$, and a set of users $\mathcal{U}$ of size t, compute $m_i := \lambda_i s u$ using Lagrange coefficient λ_i.

Sample noise $E_i \leftarrow R_q$ s.t $\|E_i\|_\infty \leq 2^{\mathsf{sec}} B_{\mathsf{Dec}}$, then output $\mathsf{ds}_i := m_i + pE_i$.

Comb On input a ciphertext $\mathsf{ctx} = (u, v)$ and a set of partial decryption shares $\{\mathsf{ds}_j\}_{j \in \mathcal{U}}$, it outputs $m := (v - \sum_{j \in \mathcal{U}} \mathsf{ds}_j) \mod p$.

BGV TDec

The threshold decryption procedure for BGV works as follows:

TDec On input a ciphertext $\mathsf{ctx} = (u, v)$, a decryption key share $\mathsf{sk}_i = s_i$, and a set of users $\mathcal{U}$ of size t, compute $m_i := \lambda_i s u$ using Lagrange coefficient λ_i.

Sample noise $E_i \leftarrow R_q$ s.t $\|E_i\|_\infty \leq 2^{\mathsf{sec}} B_{\mathsf{Dec}}$, then output $\mathsf{ds}_i := m_i + pE_i$.

Comb On input a ciphertext $\mathsf{ctx} = (u, v)$ and a set of partial decryption shares $\{\mathsf{ds}_j\}_{j \in \mathcal{U}}$, it outputs $m := (v - \sum_{j \in \mathcal{U}} \mathsf{ds}_j) \mod p$.

- $\rightarrow$ TDec can be made actively secure using commitments and ZKPs.

Contents

Ideas

Ideas

- Use noise drowning techniques to avoid rejection sampling

Ideas

- Use noise drowning techniques to avoid rejection sampling
- Use linearly homomorphic encryption to combine shares

Ideas

- Use noise drowning techniques to avoid rejection sampling
- Use linearly homomorphic encryption to combine shares
- Use t-out-of-n threshold decryption to reconstruct signatures

Passive Protocol

Keys $\mathbf{s}$ and $(\bar{\mathbf{A}}, \boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s})$ are as before. Instead of sharing $\mathbf{s}$, signers will hold an encryption $\mathsf{ctx}_{\mathbf{s}} = \mathsf{Enc}(\mathbf{s})$ and share the decryption key $\boldsymbol{k}$ in a t-out-of-n fashion:

Passive Protocol

Keys $\mathbf{s}$ and $(\bar{\mathbf{A}}, \boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s})$ are as before. Instead of sharing $\mathbf{s}$, signers will hold an encryption $\mathsf{ctx}_{\mathbf{s}} = \mathsf{Enc}(\mathbf{s})$ and share the decryption key $\boldsymbol{k}$ in a t-out-of-n fashion:

1. The ith signer chooses a short vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$. It also sends $\mathsf{ctx}_{\boldsymbol{r}_i}$, an encryption of $\boldsymbol{r}_i$.

Passive Protocol

Keys $\mathbf{s}$ and $(\bar{\mathbf{A}}, \boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s})$ are as before. Instead of sharing $\mathbf{s}$, signers will hold an encryption $\mathsf{ctx}_{\mathbf{s}} = \mathsf{Enc}(\mathbf{s})$ and share the decryption key $\boldsymbol{k}$ in a t-out-of-n fashion:

1. The ith signer chooses a short vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$. It also sends $\mathsf{ctx}_{\boldsymbol{r}_i}$, an encryption of $\boldsymbol{r}_i$.
2. Each signer computes $\boldsymbol{w} := \sum_{i \in \mathcal{U}} \boldsymbol{w}_i$, $c = H(\boldsymbol{w})$, and "encrypted signature" $\mathsf{ctx}_{\boldsymbol{z}} := c \cdot \mathsf{ctx}_{\mathbf{s}} + \sum_{i \in \mathcal{U}} \mathsf{ctx}_{\boldsymbol{r}_i}$. It sends its threshold decryption share of $\mathsf{ctx}_{\boldsymbol{z}}$.

Passive Protocol

Keys $\mathbf{s}$ and $(\bar{\mathbf{A}}, \boldsymbol{y} := \bar{\mathbf{A}}\mathbf{s})$ are as before. Instead of sharing $\mathbf{s}$, signers will hold an encryption $\mathsf{ctx}_{\mathbf{s}} = \mathsf{Enc}(\mathbf{s})$ and share the decryption key $\boldsymbol{k}$ in a t-out-of-n fashion:

1. The ith signer chooses a short vector $\boldsymbol{r}_i \in R_q^{\ell+k}$ and sends $\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i$. It also sends $\mathsf{ctx}_{\boldsymbol{r}_i}$, an encryption of $\boldsymbol{r}_i$.
2. Each signer computes $\boldsymbol{w} := \sum_{i \in \mathcal{U}} \boldsymbol{w}_i$, $c = H(\boldsymbol{w})$, and "encrypted signature" $\mathsf{ctx}_{\boldsymbol{z}} := c \cdot \mathsf{ctx}_{\mathbf{s}} + \sum_{i \in \mathcal{U}} \mathsf{ctx}_{\boldsymbol{r}_i}$. It sends its threshold decryption share of $\mathsf{ctx}_{\boldsymbol{z}}$.
3. Given decryption shares from all parties, each signer can decrypt $\mathsf{ctx}_{\boldsymbol{z}}$ to obtain $\boldsymbol{z}$, and output the signature $(c, \boldsymbol{z})$.

Passive Protocol

$\underline{\mathsf{Sign}_{\mathcal{TS}}(\mathsf{sk}_i, \mathcal{U}, \mu)}$

sample bounded $\boldsymbol{r}_i \leftarrow D_r$

$\boldsymbol{w}_i := \bar{\mathbf{A}}\boldsymbol{r}_i, \quad \mathsf{ctx}_{\boldsymbol{r}_i} := \mathsf{Enc}(\mathsf{pk}_{\mathcal{E}}, \boldsymbol{r}_i) \qquad \xrightarrow{\boldsymbol{w}_i,\, \mathsf{ctx}_{\boldsymbol{r}_i}}$

$\boldsymbol{w} := \sum_{j \in \mathcal{U}} \boldsymbol{w}_j, \quad c := H(\boldsymbol{w}, \mathsf{pk}, \mu) \qquad \xleftarrow{\{(\boldsymbol{w}_j,\, \mathsf{ctx}_{\boldsymbol{r}_j})\}_{j \in \mathcal{U} \setminus \{i\}}}$

$\mathsf{ctx}_{\mathbf{z}} := c \cdot \mathsf{ctx}_{\mathbf{s}} + \sum_{j \in \mathcal{U}} \mathsf{ctx}_{\boldsymbol{r}_j}$

$\mathsf{ds}_i := \mathsf{TDec}(\mathsf{ctx}_{\mathbf{z}}, \mathsf{sk}_i, \mathcal{U}) \qquad \xrightarrow{\mathsf{ds}_i}$

$\boldsymbol{z} := \mathsf{Comb}(\mathsf{ctx}_{\mathbf{z}}, \{\mathsf{ds}_j\}_{j \in \mathcal{U}}) \qquad \xleftarrow{\{\mathsf{ds}_j\}_{j \in \mathcal{U} \setminus \{i\}}}$

return $\sigma := (c, \boldsymbol{z})$

Contents

Setting

- Signing threshold of $t = 3$ out of $n = 5$ signers
- Signing at most 1 or 365 or 2^{64} signatures total
- Focus on signature and key size, not communication

Performance Estimates

Comm.	1-SIG	1-PK	365-SIG	365-PK	∞-SIG	∞-PK
Size	8.5 KB	2.6 KB	10.4 KB	3.1 KB	46.6 KB	13.6 KB

Approximate sizes for 3-out-of-5 threshold signatures with trusted setup.

Ongoing Work

Ongoing Work

- Use modules lattices instead of plain rings for a more flexible design

Ongoing Work

- Use modules lattices instead of plain rings for a more flexible design
- Utilize compression techniques to improve size

Ongoing Work

- Use modules lattices instead of plain rings for a more flexible design
- Utilize compression techniques to improve size
- Instantiate the distributed key generation protocol as well

Ongoing Work

- Use modules lattices instead of plain rings for a more flexible design
- Utilize compression techniques to improve size
- Instantiate the distributed key generation protocol as well
- Detail the communication and optimize parameters and proofs

Ongoing Work

- Use modules lattices instead of plain rings for a more flexible design
- Utilize compression techniques to improve size
- Instantiate the distributed key generation protocol as well
- Detail the communication and optimize parameters and proofs
- Implement the scheme for more thresholds and signature bounds

Ongoing Work

- Use modules lattices instead of plain rings for a more flexible design
- Utilize compression techniques to improve size
- Instantiate the distributed key generation protocol as well
- Detail the communication and optimize parameters and proofs
- Implement the scheme for more thresholds and signature bounds
- Provide pre-computation and non-interactive signing

Thank you! Questions?

The paper is available at: `https://eprint.iacr.org/2023/1318`