Accountability and Privacy for Threshold Signatures

Dan Boneh[1] Chelsea Komlo[2]

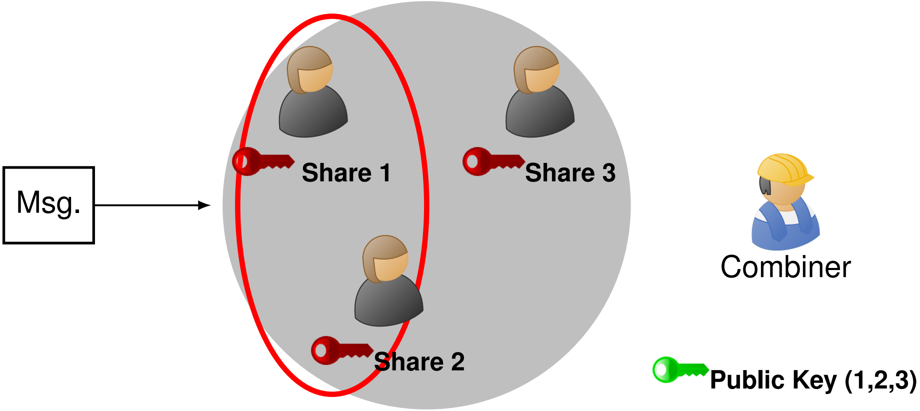
Stanford University

University of Waterloo

University of St. Gallen, November 16, 2022

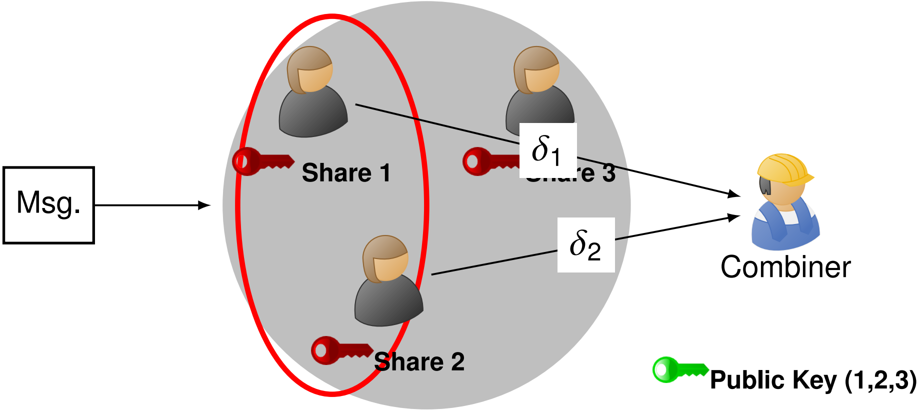
Threshold Signatures: Joint Public Key, Secret-Shared Private Key

$(t = 2, n = 3)$ Example



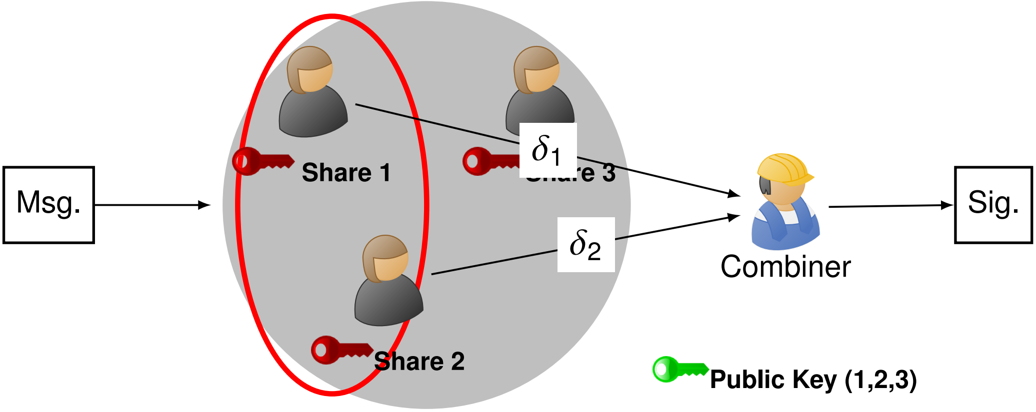
Threshold Signatures: Joint Public Key, Secret-Shared Private Key

($t = 2, n = 3$) Example



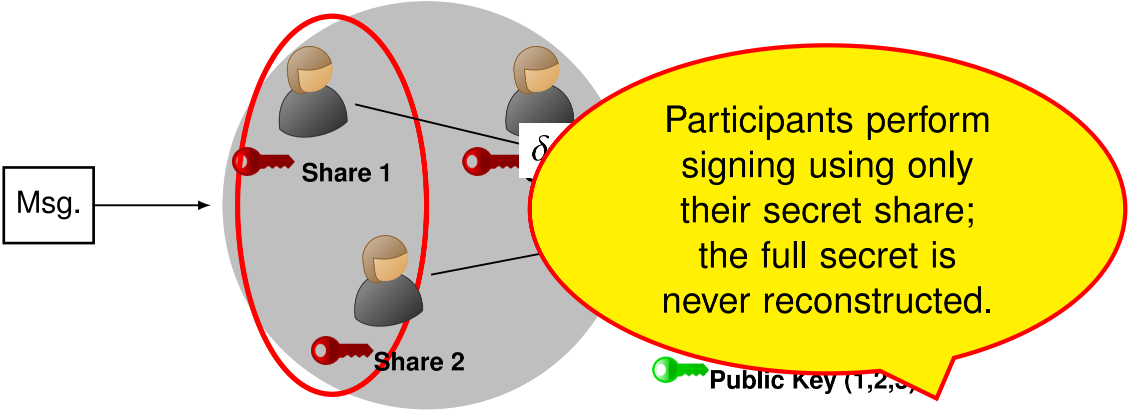
Threshold Signatures: Joint Public Key, Secret-Shared Private Key

$(t = 2, n = 3)$ Example



Threshold Signatures: Joint Public Key, Secret-Shared Private Key

$(t = 2, n = 3)$ Example



Existing Threshold Schemes

- In the literature, two types of (mutually exclusive) threshold schemes.
- **Full Privacy**: Private Threshold Scheme (PTS)
- **Full Accountability**: Accountable Threshold Scheme (ATS).
  - Also known as an Accountable Subgroup Multisignatures

Existing Threshold Schemes

- In the literature, two types of (mutually exclusive) threshold schemes.
- **Full Privacy**: Private Threshold Scheme (PTS)
- **Full Accountability**: Accountable Threshold Scheme (ATS).
  - Also known as an Accountable Subgroup Multisignatures

Existing Threshold Schemes

- In the literature, two types of (mutually exclusive) threshold schemes.
- **Full Privacy**: Private Threshold Scheme (PTS)
- **Full Accountability**: Accountable Threshold Scheme (ATS).
  - Also known as an Accountable Subgroup Multisignatures

Existing Threshold Schemes

- In the literature, two types of (mutually exclusive) threshold schemes.
- **Full Privacy**: Private Threshold Scheme (PTS)
- **Full Accountability**: Accountable Threshold Scheme (ATS).
  - Also known as an Accountable Subgroup Multisignatures

Definition

A **private threshold signature** scheme, or **PTS**, is a tuple of four polynomial time algorithms

$$\mathcal{S} = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Combine}, \mathit{Verify})$$

Definition

An **accountable threshold signature** scheme, or **ATS**, is a tuple of five polynomial time algorithms

$$\mathcal{S} = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Combine}, \mathit{Verify}, \mathit{Trace})$$

Such that $\mathit{Trace}(pk, m, \sigma) \rightarrow C \subseteq \{1, \ldots, n\}$

PTS versus ATS

	who learns signer quorum		**who learns threshold**	
	public	signers	public	signers
PTS	X	X	X	✓
ATS	✓	✓	✓	✓

Segway: Schnorr PTS/ATS Schemes

Let's see how existing multi-party Schnorr signature schemes meet these PTS/ATS notions.

Single-Party Schnorr Signing and Verification

Signer		**Verifier**
$(x, Y) \leftarrow KeyGen()$		
	$\xleftarrow{(m, Y)}$	
$k \xleftarrow{\$} \mathbb{Z}_q$ $R = g^k \in \mathbb{G}$ $c = H(R, Y, m)$ $z = k + c \cdot x$		
	$\xrightarrow{(m, \sigma = (R, z))}$	
		$c = H(R, Y, m)$ $R' = g^z \cdot Y^{-c}$ Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer **Verifier**

$(x, Y) \leftarrow KeyGen()$

$\xleftarrow{\quad (m, Y) \quad}$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$\xrightarrow{\quad (m, \sigma = (R, z)) \quad}$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer

$(x, Y) \leftarrow KeyGen()$

Verifier

$$\xleftarrow{(m, Y)}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{(m, \sigma = (R, z))}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer | **Verifier**

$(x, Y) \leftarrow KeyGen()$

$$\xleftarrow{(m, Y)}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{(m, \sigma = (R, z))}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer **Verifier**

$(x, Y) \leftarrow KeyGen()$

$$\xleftarrow{(m, Y)}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{(m, \sigma = (R, z))}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer **Verifier**

$(x, Y) \leftarrow KeyGen()$

$$\xleftarrow{\quad (m, Y) \quad}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{\quad (m, \sigma = (R, z)) \quad}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer **Verifier**

$(x, Y) \leftarrow KeyGen()$

$$\xleftarrow{\quad (m, Y) \quad}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{\quad (m, \sigma = (R, z)) \quad}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer | **Verifier**

$(x, Y) \leftarrow KeyGen()$

$$\xleftarrow{(m, Y)}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{(m, \sigma = (R, z))}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer | **Verifier**

$(x, Y) \leftarrow KeyGen()$

$$\xleftarrow{\quad (m, Y) \quad}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{\quad (m, \sigma = (R, z)) \quad}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Single-Party Schnorr Signing and Verification

Signer | **Verifier**

$(x, Y) \leftarrow KeyGen()$

$$\xleftarrow{\quad (m, Y) \quad}$$

$k \xleftarrow{\$} \mathbb{Z}_q$

$R = g^k \in \mathbb{G}$

$c = H(R, Y, m)$

$z = k + c \cdot x$

$$\xrightarrow{\quad (m, \sigma = (R, z)) \quad}$$

$c = H(R, Y, m)$

$R' = g^z \cdot Y^{-c}$

Output $R \stackrel{?}{=} R'$

Example PTS: FROST

- Threshold signatures that rely on *polynomial interpolation* (i.e, Shamir's secret sharing) are inherently a PTS.

$$z = \sum_{i \in S} z_i = r + c \cdot \sum_{i \in S} \lambda_i \cdot sk_i$$

- Shamir's secret sharing: each $(i, sk_i), i \in \{1, \ldots, n\}$ is a point on a secret polynomial f, where $(0, f(0))$ is sk. So:

$$sk = \sum_{i \in S} \lambda_i \cdot sk_i$$

- Where λ_i is the i^{th} Lagrange coefficient.

Example PTS: FROST

- Threshold signatures that rely on *polynomial interpolation* (i.e, Shamir's secret sharing) are inherently a PTS.

$$z = \sum_{i \in S} z_i = r + c \cdot \sum_{i \in S} \lambda_i \cdot sk_i$$

- Shamir's secret sharing: each $(i, sk_i), i \in \{1, \ldots, n\}$ is a point on a secret polynomial f, where $(0, f(0))$ is sk. So:

$$sk = \sum_{i \in S} \lambda_i \cdot sk_i$$

- Where λ_i is the i^{th} Lagrange coefficient.

Example PTS: FROST

- Threshold signatures that rely on *polynomial interpolation* (i.e, Shamir's secret sharing) are inherently a PTS.

$$z = \sum_{i \in S} z_i = r + c \cdot \sum_{i \in S} \lambda_i \cdot sk_i$$

- Shamir's secret sharing: each $(i, sk_i), i \in \{1, \ldots, n\}$ is a point on a secret polynomial f, where $(0, f(0))$ is sk. So:

$$sk = \sum_{i \in S} \lambda_i \cdot sk_i$$

- Where λ_i is the i^{th} Lagrange coefficient.

Example PTS: FROST

- Threshold signatures that rely on *polynomial interpolation* (i.e, Shamir's secret sharing) are inherently a PTS.

$$z = \sum_{i \in S} z_i = r + c \cdot \sum_{i \in S} \lambda_i \cdot sk_i$$

- Shamir's secret sharing: each $(i, sk_i), i \in \{1, \ldots, n\}$ is a point on a secret polynomial f, where $(0, f(0))$ is sk. So:

$$sk = \sum_{i \in S} \lambda_i \cdot sk_i$$

- Where λ_i is the i^{th} Lagrange coefficient.

Example PTS: FROST

- Threshold signatures that rely on *polynomial interpolation* (i.e, Shamir's secret sharing) are inherently a PTS.

$$z = \sum_{i \in S} z_i = r + c \cdot \sum_{i \in S} \lambda_i \cdot sk_i$$

- Shamir's secret sharing: each $(i, sk_i), i \in \{1, \ldots, n\}$ is a point on a secret polynomial f, where $(0, f(0))$ is sk. So:

$$sk = \sum_{i \in S} \lambda_i \cdot sk_i$$

- Where λ_i is the i^{th} Lagrange coefficient.

Example ATS: MuSig2

- Key generation: All parties perform independently, where:

$$pk = \{t, (pk_1, \ldots, pk_n)\}$$

- Signature algorithm is similar to FROST.

$$z = \sum_{i \in S} z_i = r + c \cdot \sum_{i \in S} \alpha_i \cdot sk_i$$

- Where $\alpha_i = H(S, pk_i)$.

- The signature is valid with respect to the product of t of public keys in pk (and so signers can be identified).

Example ATS: MuSig2

- Key generation: All parties perform independently, where:

$$pk = \{t, (pk_1, \ldots, pk_n)\}$$

- Signature algorithm is similar to FROST.

$$z = \sum_{i \in S} z_i = r + c \cdot \sum_{i \in S} \alpha_i \cdot sk_i$$

- Where $\alpha_i = H(S, pk_i)$.
- The signature is valid with respect to the product of t of public keys in pk (and so signers can be identified).

Example ATS: MuSig2

- Key generation: All parties perform independently, where:

$$pk = \{t, (pk_1, \ldots, pk_n)\}$$

- Signature algorithm is similar to FROST.

$$z = \sum_{i \in S} z_i = r + c \cdot \sum_{i \in S} \alpha_i \cdot sk_i$$

- Where $\alpha_i = H(S, pk_i)$.

- The signature is valid with respect to the product of t of public keys in pk (and so signers can be identified).

Example ATS: MuSig2

- Key generation: All parties perform independently, where:

$$pk = \{t, (pk_1, \ldots, pk_n)\}$$

- Signature algorithm is similar to FROST.

$$z = \sum_{i \in S} z_i = r + c \cdot \sum_{i \in S} \alpha_i \cdot sk_i$$

- Where $\alpha_i = H(S, pk_i)$.

- The signature is valid with respect to the product of t of public keys in pk (and so signers can be identified).

Example ATS: MuSig2

- Key generation: All parties perform independently, where:

$$pk = \{t, (pk_1, \ldots, pk_n)\}$$

- Signature algorithm is similar to FROST.

$$z = \sum_{i \in S} z_i = r + c \cdot \sum_{i \in S} \alpha_i \cdot sk_i$$

- Where $\alpha_i = H(S, pk_i)$.

- The signature is valid with respect to the product of t of public keys in pk (and so signers can be identified).

TAPS: Privacy and Accountability

- We introduce a *new* type of threshold signature.
- **Private *and* Accountable Threshold Signatures** (TAPS)
- Like a PTS, the public does not learn 1) the threshold, or 2) the signing quorum
- Like an ATS, the signing quorum can be recovered, but *only* by a designated entity.

TAPS: Privacy and Accountability

- We introduce a *new* type of threshold signature.
- **Private *and* Accountable Threshold Signatures** (TAPS)
- Like a PTS, the public does not learn 1) the threshold, or 2) the signing quorum
- Like an ATS, the signing quorum can be recovered, but *only* by a designated entity.

TAPS: Privacy and Accountability

- We introduce a *new* type of threshold signature.
- **Private *and* Accountable Threshold Signatures** (TAPS)
- Like a PTS, the public does not learn 1) the threshold, or 2) the signing quorum
- Like an ATS, the signing quorum can be recovered, but *only* by a designated entity.

TAPS: Privacy and Accountability

- We introduce a *new* type of threshold signature.
- **Private *and* Accountable Threshold Signatures** (TAPS)
- Like a PTS, the public does not learn 1) the threshold, or 2) the signing quorum
- Like an ATS, the signing quorum can be recovered, but *only* by a designated entity.

TAPS: Private and Accountable Threshold Signatures

Applications include:

- **Financial institutions**: prove or disprove issuance of funds.
- **Post-Compromise Accountability**: Allows for identification of malicious signers.

TAPS: Private and Accountable Threshold Signatures

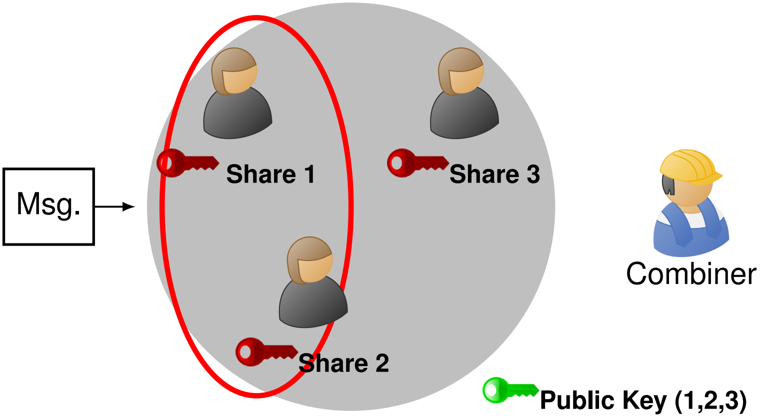
Applications include:

- **Financial institutions**: prove or disprove issuance of funds.

- **Post-Compromise Accountability**: Allows for identification of malicious signers.

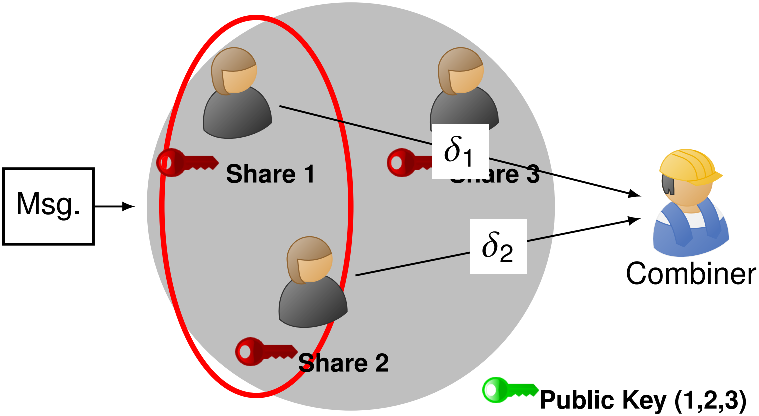
TAPS

$(t = 2, n = 3)$ Example



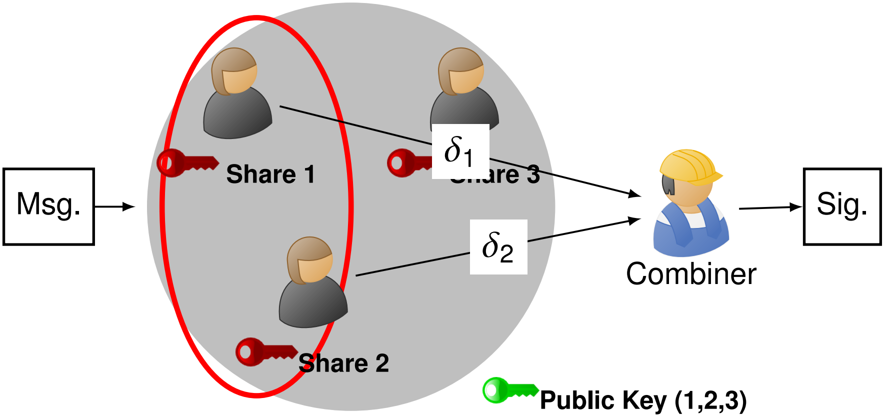
TAPS

$(t = 2, n = 3)$ Example



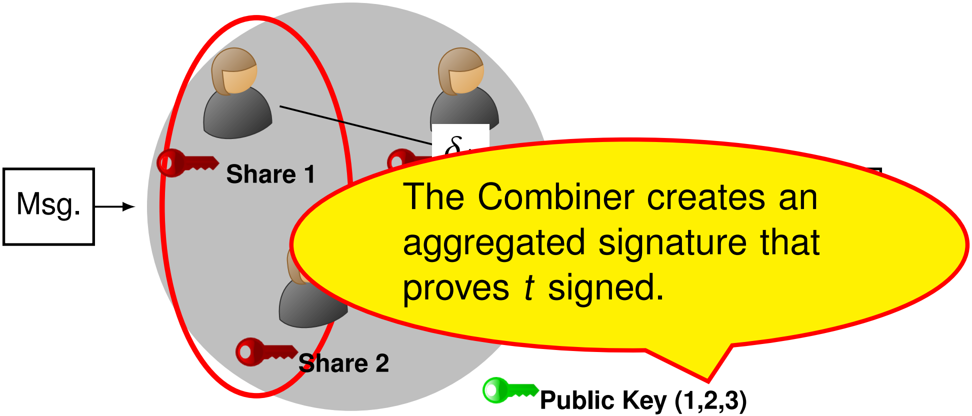
TAPS

$(t = 2, n = 3)$ Example



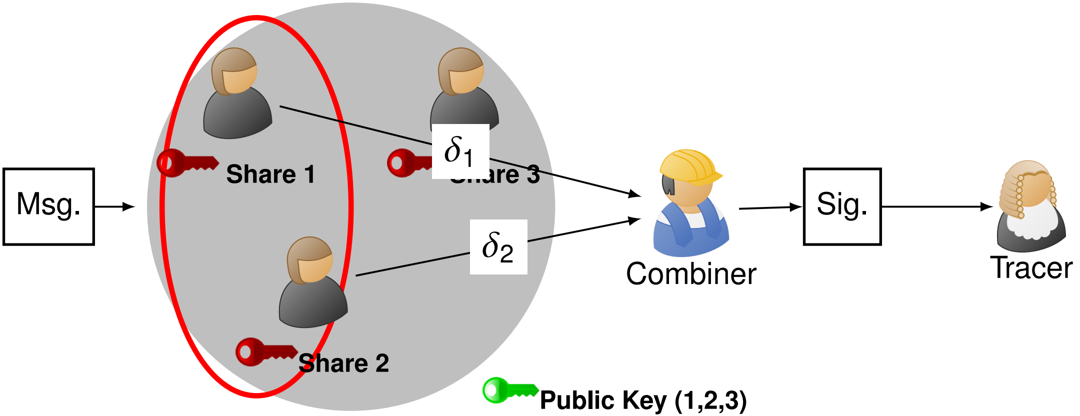
TAPS

$(t = 2, n = 3)$ Example



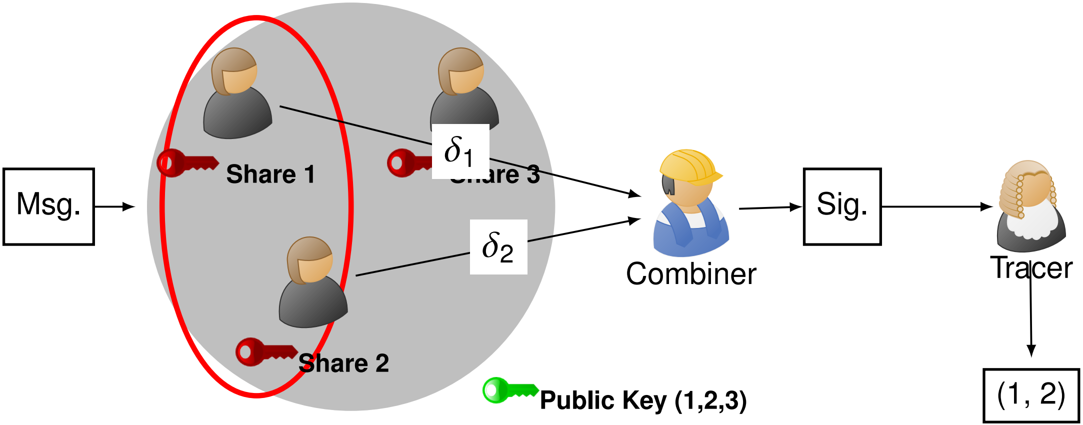
TAPS

($t = 2, n = 3$) Example



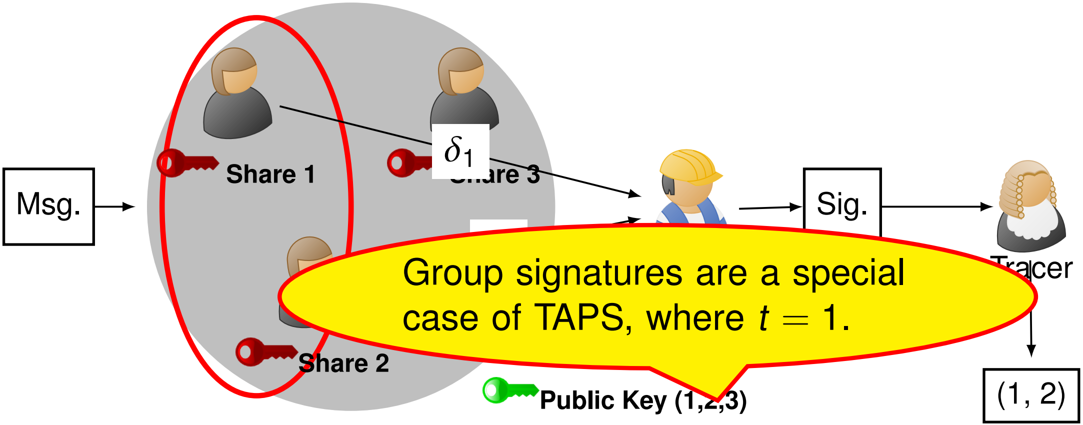
TAPS

$(t = 2, n = 3)$ Example



TAPS

$(t = 2, n = 3)$ Example



TAPS

Definition

A **private and accountable threshold signature** scheme, or **TAPS**, is a tuple of five polynomial time algorithms

$$\mathcal{S} = (\mathit{KeyGen}, \mathit{Sign}, \mathit{Combine}, \mathit{Verify}, \mathit{Trace})$$

Where *Trace* can be performed only by a designated entity.

A TAPS must be secure (unforgeable), private, and accountable.

TAPS

$$KeyGen(1^\lambda, n, t) \rightarrow \left(pk, (sk_1, \ldots, sk_n), sk_c, sk_t\right)$$

- pk: The group's public key
- $(sk_1, \ldots, sk_n)$: Secret keys for each of the n participants.
- sk_c: Secret key for the combiner
- sk_t: Secret key for the tracer

- Could be performed in a distributed manner.

TAPS

$$KeyGen(1^\lambda, n, t) \rightarrow \left(pk, (sk_1, \ldots, sk_n), sk_{\mathrm{c}}, sk_{\mathrm{t}}\right)$$

- pk: The group's public key
- $(sk_1, \ldots, sk_n)$: Secret keys for each of the n participants.
- sk_{c}: Secret key for the combiner
- sk_{t}: Secret key for the tracer

- Could be performed in a distributed manner.

TAPS

$$KeyGen(1^\lambda, n, t) \rightarrow \big(pk, (sk_1, \ldots, sk_n), sk_{\mathrm{c}}, sk_{\mathrm{t}}\big)$$

- pk: The group's public key
- $(sk_1, \ldots, sk_n)$: Secret keys for each of the n participants.
- sk_{c}: Secret key for the combiner
- sk_{t}: Secret key for the tracer

- Could be performed in a distributed manner.

TAPS

$$KeyGen(1^\lambda, n, t) \rightarrow \left(pk, (sk_1, \ldots, sk_n), sk_\mathrm{c}, sk_\mathrm{t}\right)$$

- pk: The group's public key
- $(sk_1, \ldots, sk_n)$: Secret keys for each of the n participants.
- sk_c: Secret key for the combiner
- sk_t: Secret key for the tracer

- Could be performed in a distributed manner.

TAPS

$$KeyGen(1^\lambda, n, t) \rightarrow \left(pk, (sk_1, \ldots, sk_n), sk_{\mathrm{c}}, sk_{\mathrm{t}}\right)$$

- pk: The group's public key
- $(sk_1, \ldots, sk_n)$: Secret keys for each of the n participants.
- sk_{c}: Secret key for the combiner
- sk_{t}: Secret key for the tracer

- Could be performed in a distributed manner.

TAPS

$Sign(sk_i, m, C) \rightarrow \delta_i$

- m: Message to be signed
- C: Coalition of signers
- δ_i: Partial signature for participant i

$Combine(sk_c, m, C, \{\delta_i\}_{i \in C}) \rightarrow \sigma$

- Outputs σ, a TAPS signature

TAPS

$Sign(sk_i, m, C) \rightarrow \delta_i$

- m: Message to be signed
- C: Coalition of signers
- δ_i: Partial signature for participant i

$Combine(sk_c, m, C, \{\delta_i\}_{i \in C}) \rightarrow \sigma$

- Outputs σ, a TAPS signature

TAPS

$Sign(sk_i, m, C) \rightarrow \delta_i$

- m: Message to be signed
- C: Coalition of signers
- δ_i: Partial signature for participant i

$Combine(sk_c, m, C, \{\delta_i\}_{i \in C}) \rightarrow \sigma$

- Outputs σ, a TAPS signature

TAPS

$Sign(sk_i, m, C) \rightarrow \delta_i$

- m: Message to be signed
- C: Coalition of signers
- δ_i: Partial signature for participant i

$Combine(sk_c, m, C, \{\delta_i\}_{i \in C}) \rightarrow \sigma$

- Outputs σ, a TAPS signature

TAPS

$Sign(sk_i, m, C) \rightarrow \delta_i$

- m: Message to be signed
- C: Coalition of signers
- δ_i: Partial signature for participant i

$Combine(sk_{\mathrm{c}}, m, C, \{\delta_i\}_{i \in C}) \rightarrow \sigma$

- Outputs σ, a TAPS signature

TAPS

$Verify(pk, m, \sigma) \rightarrow 0/1$:

- Outputs a bit indicating if σ is valid for pk, m

$Trace(sk_t, m, \sigma) \rightarrow C/\text{fail}$:

- Outputs either the coalition of signers or fails.

TAPS

$\mathit{Verify}(pk, m, \sigma) \rightarrow 0/1$:

- Outputs a bit indicating if σ is valid for pk, m

$\mathit{Trace}(sk_{\mathrm{t}}, m, \sigma) \rightarrow C/\text{fail}$:

- Outputs either the coalition of signers or fails.

TAPS

$Verify(pk, m, \sigma) \rightarrow 0/1$:

- Outputs a bit indicating if σ is valid for pk, m

$Trace(sk_{\mathrm{t}}, m, \sigma) \rightarrow C/\text{fail}$:

- Outputs either the coalition of signers or fails.

Unforgeability and Accountability

- **Unforgeability.** Adversary cannot output a valid signature when controlling fewer than t parties.
- **Accountability.** Adversary cannot output a valid signature that traces to an honest non-signer.

Unforgeability and Accountability

- **Unforgeability.** Adversary cannot output a valid signature when controlling fewer than t parties.
- **Accountability.** Adversary cannot output a valid signature that traces to an honest non-signer.

Unforgeability and Accountability

Adv.

$\xrightarrow{(n, t, C)}$

$\xleftarrow{(pk, \{sk_i\}_{i \in C})}$

$\xrightarrow{\mathcal{O}(C_j, m_j)}$

$\xleftarrow{\{Sign(sk_i, m_j, C_j)\}_{i \in C_j}}$

$\xrightarrow{(m', \sigma')}$

Env.

$(pk, \{sk_i\}_{i \in [n]}, sk_{\mathrm{c}}, pk_{\mathrm{t}}) \xleftarrow{\$} KeyGen(1^\lambda, n, t)$

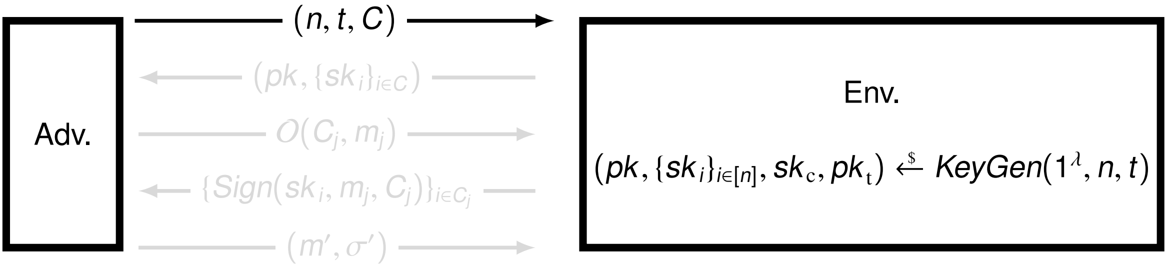
Adv wins if:

(1) It produces a valid signature and $|C| < t$ **(unforgeability)**, or

(2) It outputs a valid signature that traces to an honest non-signer $i \notin C$ **(accountability)**.

TAPS is *unforgable* and *accountable* if Pr[Adv wins] is negligible.

Unforgeability and Accountability



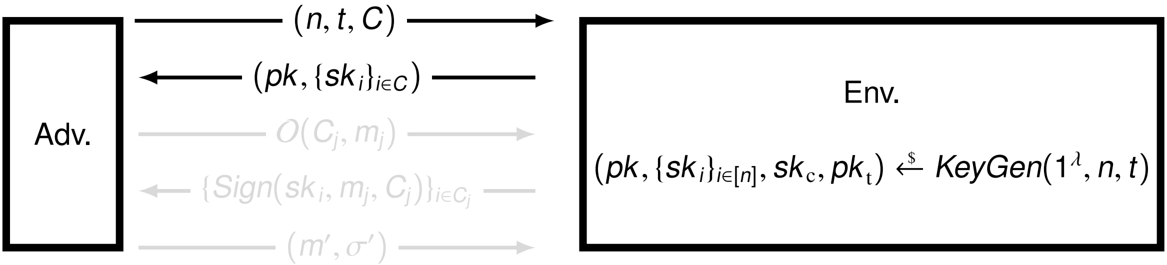
Adv wins if:

(1) It produces a valid signature and $|C| < t$ **(unforgeability)**, or

(2) It outputs a valid signature that traces to an honest non-signer $i \notin C$ **(accountability)**.

TAPS is *unforgable* and *accountable* if Pr[Adv wins] is negligible.

Unforgeability and Accountability



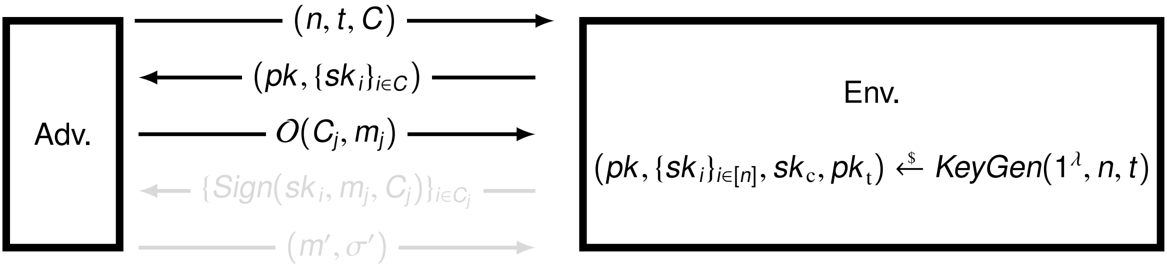
Adv wins if:

(1) It produces a valid signature and $|C| < t$ **(unforgeability)**, or

(2) It outputs a valid signature that traces to an honest non-signer $i \notin C$ **(accountability)**.

TAPS is *unforgable* and *accountable* if Pr[Adv wins] is negligible.

Unforgeability and Accountability



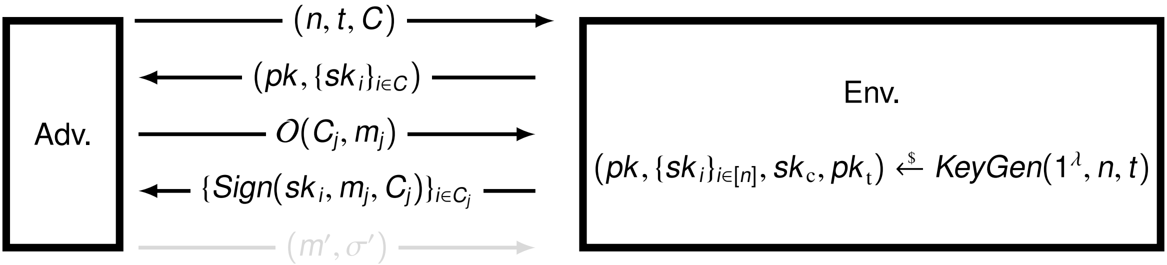
Adv wins if:

(1) It produces a valid signature and $|C| < t$ **(unforgeability)**, or

(2) It outputs a valid signature that traces to an honest non-signer $i \notin C$ **(accountability)**.

TAPS is *unforgable* and *accountable* if Pr[Adv wins] is negligible.

Unforgeability and Accountability



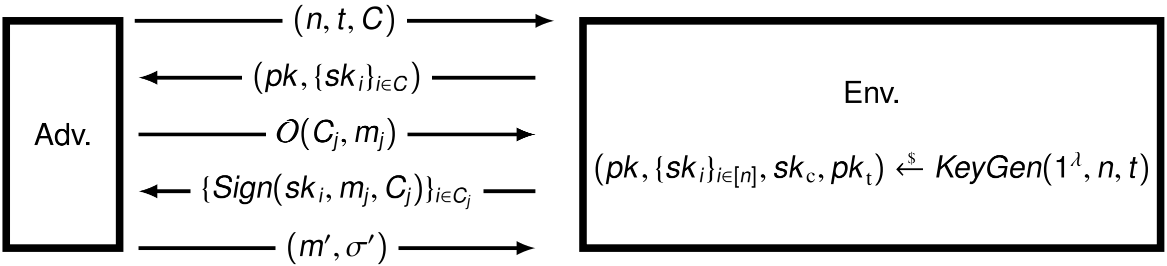
Adv wins if:

(1) It produces a valid signature and $|C| < t$ **(unforgeability)**, or

(2) It outputs a valid signature that traces to an honest non-signer $i \notin C$ **(accountability)**.

TAPS is *unforgable* and *accountable* if Pr[Adv wins] is negligible.

Unforgeability and Accountability



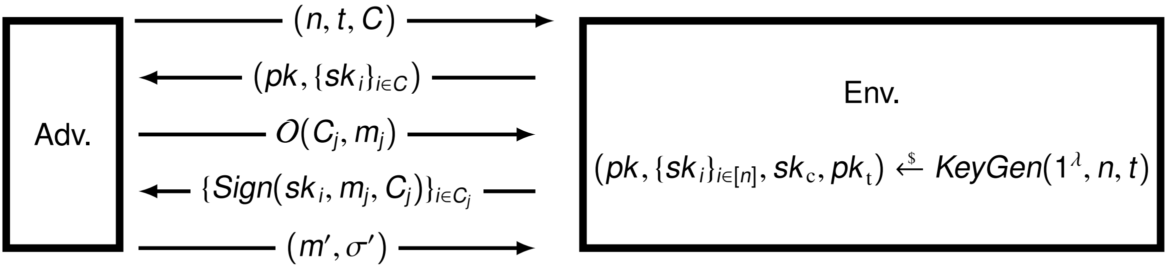
Adv wins if:

(1) It produces a valid signature and $|C| < t$ **(unforgeability)**, or

(2) It outputs a valid signature that traces to an honest non-signer $i \notin C$ **(accountability)**.

TAPS is *unforgable* and *accountable* if Pr[Adv wins] is negligible.

Unforgeability and Accountability



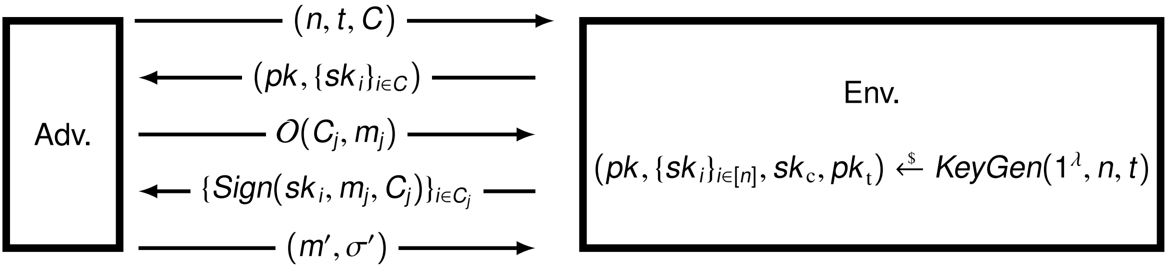
Adv wins if:

(1) It produces a valid signature and $|C| < t$ **(unforgeability)**, or

(2) It outputs a valid signature that traces to an honest non-signer $i \notin C$ **(accountability)**.

TAPS is *unforgable* and *accountable* if Pr[Adv wins] is negligible.

Unforgeability and Accountability



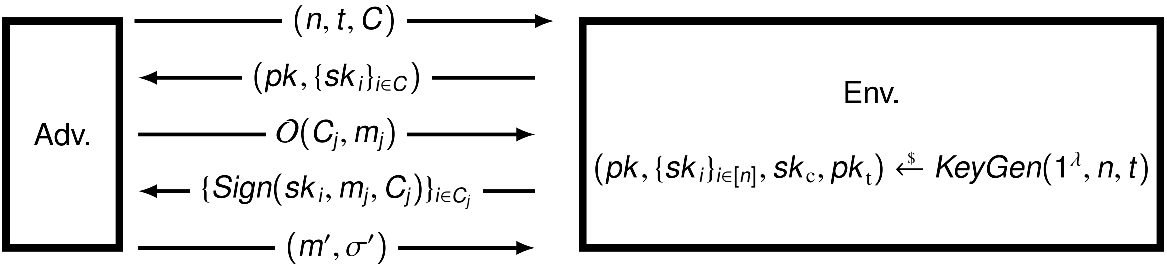
Adv wins if:

(1) It produces a valid signature and $|C| < t$ **(unforgeability)**, or

(2) It outputs a valid signature that traces to an honest non-signer $i \notin C$ **(accountability)**.

TAPS is *unforgable* and *accountable* if Pr[Adv wins] is negligible.

Unforgeability and Accountability



Adv wins if:

(1) It produces a valid signature and $|C| < t$ **(unforgeability)**, or

(2) It outputs a valid signature that traces to an honest non-signer $i \notin C$ **(accountability)**.

TAPS is *unforgable* and *accountable* if Pr[Adv wins] is negligible.

Privacy

- **Privacy against the public.** The TAPS signature reveals nothing about the threshold or quorum of signers to the *public*.
- **Privacy against signers.** The TAPS signature reveals nothing about the quorum to *signers*.

Privacy

- **Privacy against the public.** The TAPS signature reveals nothing about the threshold or quorum of signers to the *public*.

- **Privacy against signers.** The TAPS signature reveals nothing about the quorum to *signers*.

Privacy Against The Public

Adv.		Env.
	$\xrightarrow{(n, t_0, t_1)}$	
	$\xleftarrow{(pk)}$	$(pk, \{sk_i\}_{i\in[n]}, sk_c, pk_t) \xleftarrow{\$} KeyGen(1^\lambda, n, t_b)$
	$\xrightarrow{O_1(C_0, C_1, m)}$	
	$\xleftarrow{Combine(sk_c, m, C_b, \{\delta_i\}_{i\in C_b})}$	
	$\xrightarrow{O_2(m, \sigma)}$	
	$\xleftarrow{Trace(sk_t, m, \sigma)}$	
	$\xrightarrow{b'}$	

Restriction: Inputs to O_2 cannot be outputs from O_1.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against The Public

Adv.

$\xrightarrow{(n, t_0, t_1)}$

$\xleftarrow{(pk)}$

$\xrightarrow{\mathcal{O}_1(C_0, C_1, m)}$

$\xleftarrow{Combine(sk_c, m, C_b, \{\delta_i\}_{i \in C_b})}$

$\xrightarrow{\mathcal{O}_2(m, \sigma)}$

$\xleftarrow{Trace(sk_t, m, \sigma)}$

$\xrightarrow{b'}$

Env.

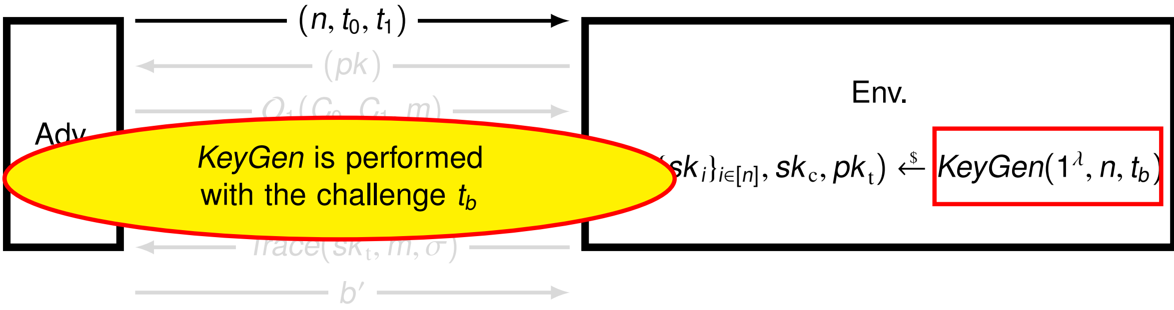
$(pk, \{sk_i\}_{i \in [n]}, sk_c, pk_t) \xleftarrow{\$} KeyGen(1^\lambda, n, t_b)$

Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against The Public



Restriction: Inputs to O_2 cannot be outputs from O_1.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against The Public

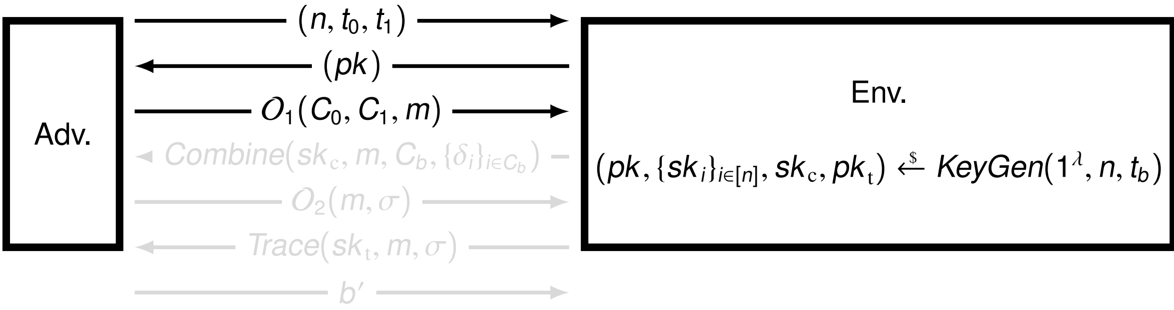
Adv.		Env.
	$\xrightarrow{(n, t_0, t_1)}$	
	$\xleftarrow{(pk)}$	$(pk, \{sk_i\}_{i\in[n]}, sk_c, pk_t) \xleftarrow{\$} KeyGen(1^\lambda, n, t_b)$
	$\xrightarrow{O_1(C_0, C_1, m)}$	
	$\xleftarrow{Combine(sk_c, m, C_b, \{\delta_i\}_{i\in C_b})}$	
	$\xrightarrow{O_2(m, \sigma)}$	
	$\xleftarrow{Trace(sk_t, m, \sigma)}$	
	$\xrightarrow{b'}$	

Restriction: Inputs to O_2 cannot be outputs from O_1.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against The Public

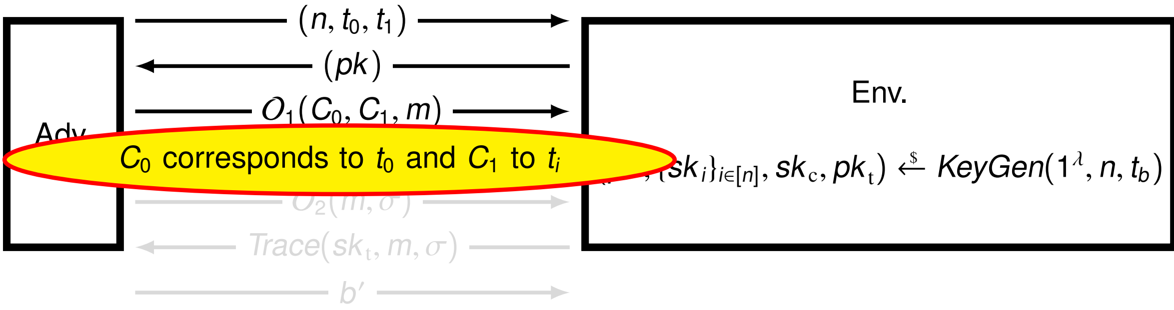


Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against The Public



Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against The Public

Adv.

$(n, t_0, t_1) \longrightarrow$

$\longleftarrow (pk)$

$\mathcal{O}_1(C_0, C_1, m) \longrightarrow$

$\longleftarrow \mathit{Combine}(sk_{\mathrm{c}}, m, C_b, \{\delta_i\}_{i \in C_b})$

$\mathcal{O}_2(m, \sigma) \longrightarrow$

$\longleftarrow \mathit{Trace}(sk_{\mathrm{t}}, m, \sigma)$

$b' \longrightarrow$

Env.

$(pk, \{sk_i\}_{i \in [n]}, sk_{\mathrm{c}}, pk_{\mathrm{t}}) \xleftarrow{\$} \mathit{KeyGen}(1^\lambda, n, t_b)$

Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against The Public

Adv.

$\xrightarrow{(n, t_0, t_1)}$

$\xleftarrow{(pk)}$

$\xrightarrow{\mathcal{O}_1(C_0, C_1, m)}$

$\xleftarrow{Combine(sk_{\mathrm{c}}, m, C_b, \{\delta_i\}_{i \in C_b})}$

$\xrightarrow{\mathcal{O}_2(m, \sigma)}$

$\xleftarrow{Trace(sk_{\mathrm{t}}, m, \sigma)}$

$\xrightarrow{b'}$

Env.

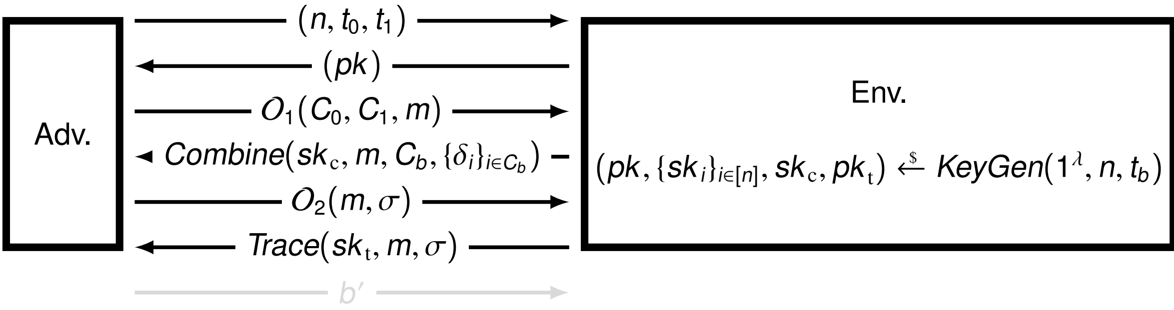
$(pk, \{sk_i\}_{i\in[n]}, sk_{\mathrm{c}}, pk_{\mathrm{t}}) \xleftarrow{\$} KeyGen(1^\lambda, n, t_b)$

Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against The Public



Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against The Public

Adv. ⟷ **Env.**

Env.: $(pk, \{sk_i\}_{i\in[n]}, sk_{\mathrm{c}}, pk_{\mathrm{t}}) \xleftarrow{\$} \mathit{KeyGen}(1^\lambda, n, t_b)$

$$\text{Adv.} \xrightarrow{(n, t_0, t_1)} \text{Env.}$$

$$\text{Adv.} \xleftarrow{(pk)} \text{Env.}$$

$$\text{Adv.} \xrightarrow{\mathcal{O}_1(C_0, C_1, m)} \text{Env.}$$

$$\text{Adv.} \xleftarrow{\mathit{Combine}(sk_{\mathrm{c}}, m, C_b, \{\delta_i\}_{i\in C_b})} \text{Env.}$$

$$\text{Adv.} \xrightarrow{\mathcal{O}_2(m, \sigma)} \text{Env.}$$

$$\text{Adv.} \xleftarrow{\mathit{Trace}(sk_{\mathrm{t}}, m, \sigma)} \text{Env.}$$

$$\text{Adv.} \xrightarrow{b'} \text{Env.}$$

Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against The Public

Adv.

$\xrightarrow{(n, t_0, t_1)}$

$\xleftarrow{(pk)}$

$\xrightarrow{\mathcal{O}_1(C_0, C_1, m)}$

$\xleftarrow{\mathsf{Combine}(sk_{\mathrm{c}}, m, C_b, \{\delta_i\}_{i \in C_b})}$

$\xrightarrow{\mathcal{O}_2(m, \sigma)}$

$\xleftarrow{\mathsf{Trace}(sk_{\mathrm{t}}, m, \sigma)}$

$\xrightarrow{b'}$

Env.

$(pk, \{sk_i\}_{i \in [n]}, sk_{\mathrm{c}}, pk_{\mathrm{t}}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, n, t_b)$

Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against the public. Adversary wins if it can gain any information about t.

TAPS is *private against the public* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers

Adv.

$(n, t) \longrightarrow$

$\longleftarrow (pk, \{sk_1, \ldots, sk_n\})$

$\mathcal{O}_1(C_0, C_1, m) \longrightarrow$

$\longleftarrow Combine(sk_c, m, C_b, \{\delta_i\}_{i \in C_b})$

$\mathcal{O}_2(m, \sigma) \longrightarrow$

$\longleftarrow Trace(sk_t, m, \sigma)$

$b' \longrightarrow$

Env.

$$(pk, \{sk_i\}_{i \in [n]}, sk_c, pk_t) \xleftarrow{\$} KeyGen(1^\lambda, n, t)$$

Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers

Adv. | **Env.**

$\xrightarrow{(n, t)}$

$(pk, \{sk_i\}_{i \in [n]}, sk_c, pk_t) \xleftarrow{\$} KeyGen(1^\lambda, n, t)$

$\xleftarrow{(pk, \{sk_1, \ldots, sk_n\})}$

$\xrightarrow{O_1(C_0, C_1, m)}$

$\xleftarrow{Combine(sk_c, m, C_b, \{\delta_i\}_{i \in C_b})}$

$\xrightarrow{O_2(m, \sigma)}$

$\xleftarrow{Trace(sk_t, m, \sigma)}$

$\xrightarrow{b'}$

Restriction: Inputs to O_2 cannot be outputs from O_1.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers

Adv. $\xrightarrow{(n, t)}$ **Env.**

Adv. $\xleftarrow{(pk, \{sk_1, \ldots, sk_n\})}$ **Env.**

Adv. $\xrightarrow{\mathcal{O}_1(C_0, C_1, m)}$ **Env.**

Adv. $\xleftarrow{Combine(sk_c, m, C_b, \{\delta_i\}_{i \in C_b})}$ **Env.**

Adv. $\xrightarrow{\mathcal{O}_2(m, \sigma)}$ **Env.**

Adv. $\xleftarrow{Trace(sk_t, m, \sigma)}$ **Env.**

Adv. $\xrightarrow{b'}$ **Env.**

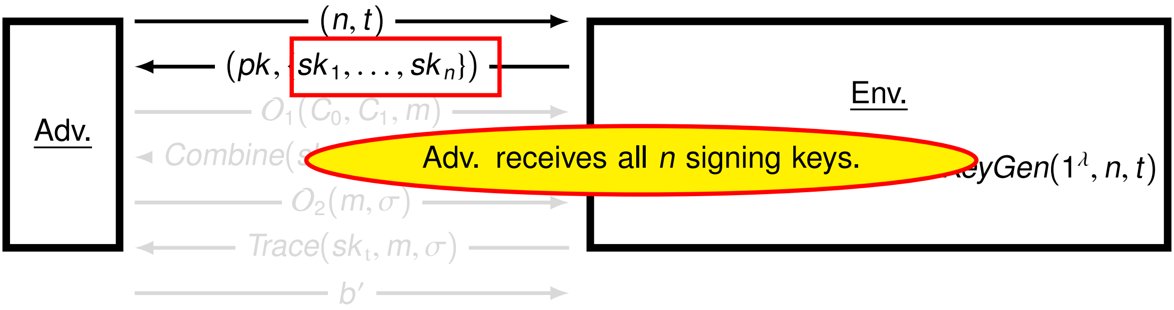
Env.: $(pk, \{sk_i\}_{i \in [n]}, sk_c, pk_t) \xleftarrow{\$} KeyGen(1^\lambda, n, t)$

Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers

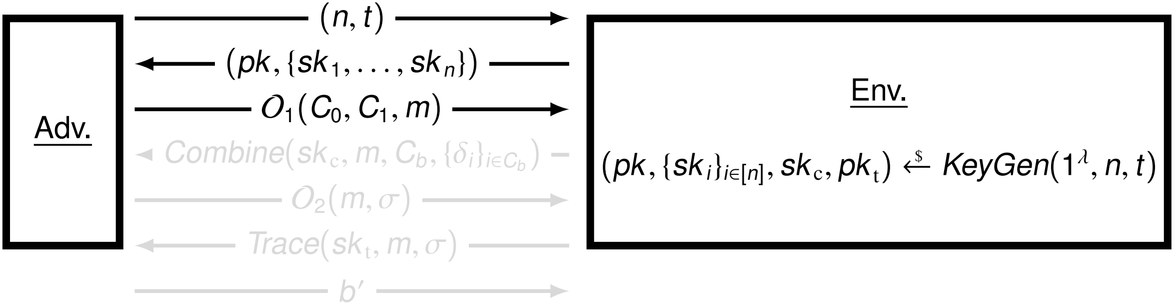


Restriction: Inputs to O_2 cannot be outputs from O_1.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers



Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers

Adv.

$\xrightarrow{(n, t)}$

$\xleftarrow{(pk, \{sk_1, \ldots, sk_n\})}$

$\xrightarrow{\mathcal{O}_1(C_0, C_1, m)}$

$\xleftarrow{Combine(sk_c, m, C_b, \{\delta_i\}_{i \in C_b})}$

$\xrightarrow{\mathcal{O}_2(m, \sigma)}$

$\xleftarrow{Trace(sk_t, m, \sigma)}$

$\xrightarrow{b'}$

Env.

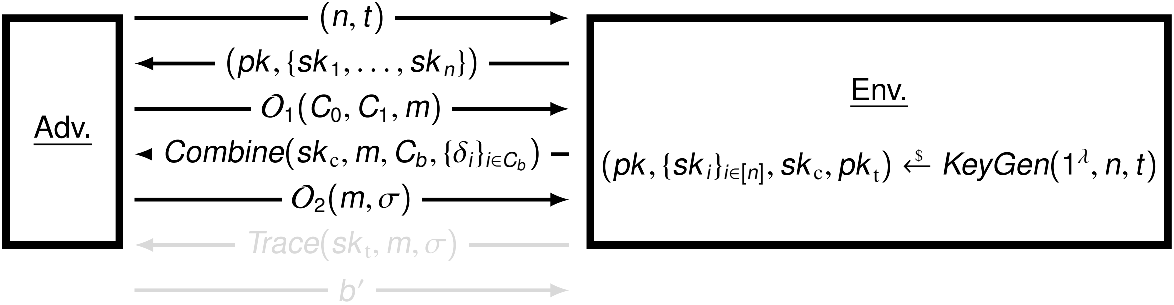
$(pk, \{sk_i\}_{i \in [n]}, sk_c, pk_t) \xleftarrow{\$} KeyGen(1^\lambda, n, t)$

Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers



Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers

Adv.

$\longrightarrow (n, t) \longrightarrow$

$\longleftarrow (pk, \{sk_1, \ldots, sk_n\}) \longleftarrow$

$\longrightarrow \mathcal{O}_1(C_0, C_1, m) \longrightarrow$

$\longleftarrow \mathit{Combine}(sk_c, m, C_b, \{\delta_i\}_{i \in C_b}) \longleftarrow$

$\longrightarrow \mathcal{O}_2(m, \sigma) \longrightarrow$

$\longleftarrow \mathit{Trace}(sk_t, m, \sigma) \longleftarrow$

$\longrightarrow b' \longrightarrow$

Env.

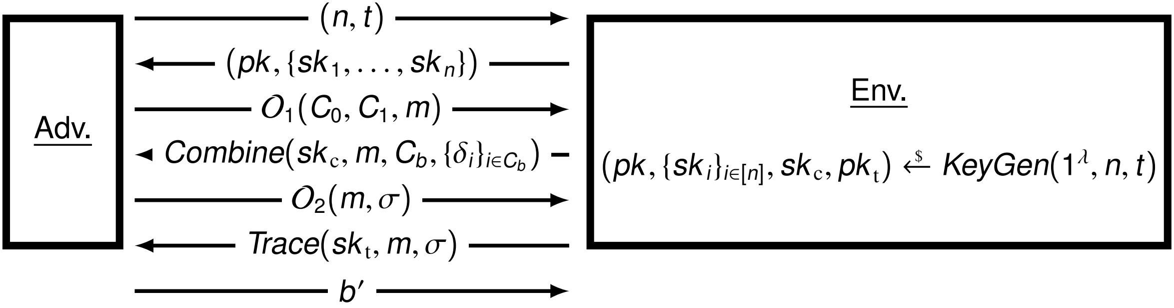
$(pk, \{sk_i\}_{i \in [n]}, sk_c, pk_t) \xleftarrow{\$} \mathit{KeyGen}(1^\lambda, n, t)$

Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers

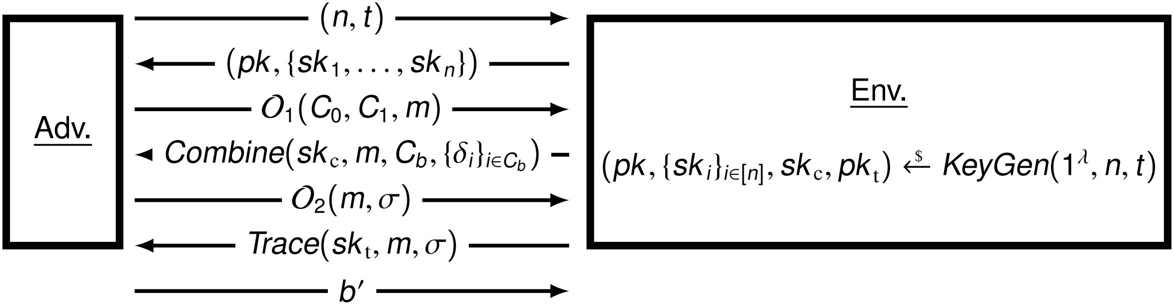


Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers

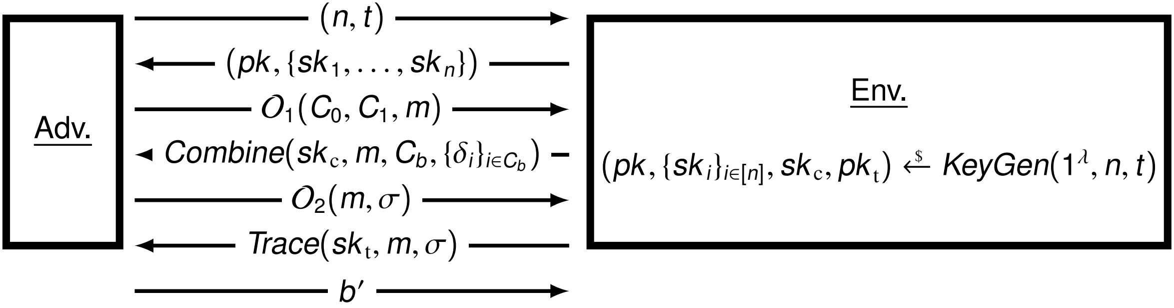


Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Privacy Against Signers



Restriction: Inputs to $\mathcal{O}_2$ cannot be outputs from $\mathcal{O}_1$.

Privacy against signers. Adv. wins if it can gain information about the quorum of signers.

TAPS is *private against signers* if Pr[Adv wins] - 1/2 is negligible.

Generic TAPS

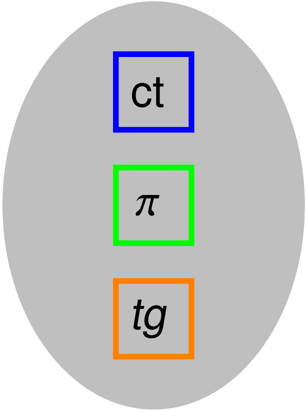
- TAPS signature σ proves in generic zero knowledge that an encrypted ATS signature is valid.
- TAPS public key contains a *commitment* to the threshold.

Generic TAPS

- TAPS signature σ proves in generic zero knowledge that an encrypted ATS signature is valid.
- TAPS public key contains a *commitment* to the threshold.

Generic TAPS: Signature

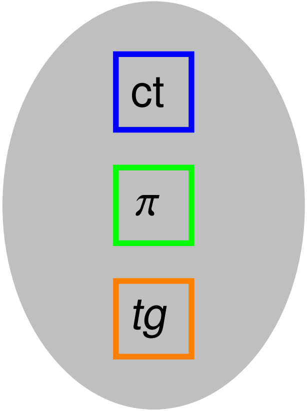
$\sigma =$



- ct: Public-key encryption of an ATS signature to the tracer.
- π: zero-knowledge proof that the ATS is a valid signature on m
- tg: The combiner's signature on (m, ct, π)

Generic TAPS: Signature

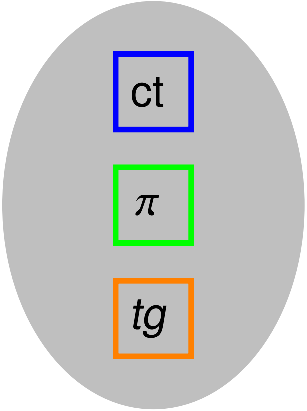
σ =



- *ct*: Public-key encryption of an ATS signature to the tracer.
- π: zero-knowledge proof that the ATS is a valid signature on m
- *tg*: The combiner's signature on (m, ct, π)

Generic TAPS: Signature

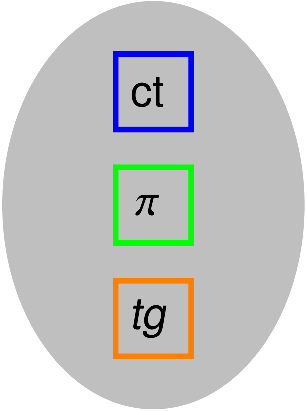
σ =



- *ct*: Public-key encryption of an ATS signature to the tracer.
- π: zero-knowledge proof that the ATS is a valid signature on *m*
- *tg*: The combiner's signature on (m, ct, π)

Generic TAPS: Signature

$\sigma =$



- *ct*: Public-key encryption of an ATS signature to the tracer.
- π: zero-knowledge proof that the ATS is a valid signature on m
- *tg*: The combiner's signature on (m, ct, π)

On Schnorr and Generic Zero Knowledge

- The generic scheme requires proving the signature is valid in generic zero knowledge.
- Schnorr requires proving the output from a hash function is derived correctly, which is expensive in generic zero knowledge.
- **Goal**: Find a simpler, more efficient solution.

On Schnorr and Generic Zero Knowledge

- The generic scheme requires proving the signature is valid in generic zero knowledge.
- Schnorr requires proving the output from a hash function is derived correctly, which is expensive in generic zero knowledge.
- **Goal**: Find a simpler, more efficient solution.

On Schnorr and Generic Zero Knowledge

- The generic scheme requires proving the signature is valid in generic zero knowledge.
- Schnorr requires proving the output from a hash function is derived correctly, which is expensive in generic zero knowledge.
- **Goal**: Find a simpler, more efficient solution.

Schnorr TAPS: Solution

- Recall a Schnorr signature σ:

$$\sigma = \begin{array}{l}(R, \quad \text{// This is a commitment to a nonce} \\ z) \quad \text{// This is the response}\end{array}$$

- The verifier derives $c = H(R, m)$, and checks $R \stackrel{?}{=} g^z \cdot pk^{-c}$.

- **Insight**: Publishing R in the clear does not hurt privacy! Only z must be protected.

Schnorr TAPS: Solution

- Recall a Schnorr signature σ:

$$\sigma = \begin{array}{l}(R, \quad /\!/ \text{ This is a commitment to a nonce} \\ z) \quad /\!/ \text{ This is the response}\end{array}$$

- The verifier derives $c = H(R, m)$, and checks $R \stackrel{?}{=} g^z \cdot pk^{-c}$.

- **Insight**: Publishing R in the clear does not hurt privacy! Only z must be protected.

Schnorr TAPS: Solution

- Recall a Schnorr signature σ:

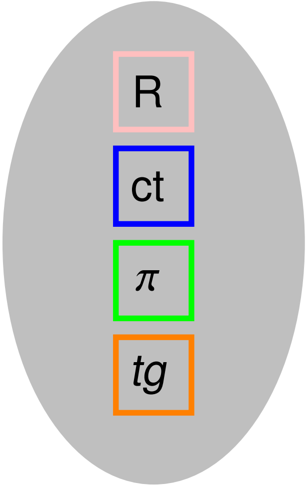
$$\sigma = \begin{array}{l} (R, \quad \text{// This is a commitment to a nonce} \\ z) \quad \text{// This is the response} \end{array}$$

- The verifier derives $c = H(R, m)$, and checks $R \stackrel{?}{=} g^z \cdot pk^{-c}$.

- **Insight**: Publishing R in the clear does not hurt privacy! Only z must be protected.

Schnorr TAPS

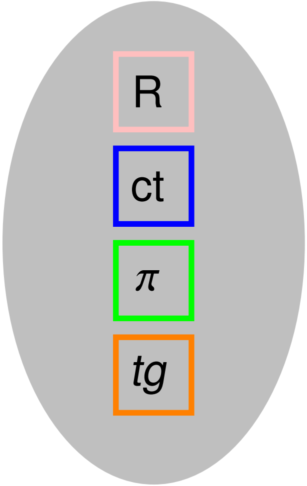
$\sigma =$



- Improved efficiency and simplicity over generic zero knowledge.
- Verifier derives the challenge directly.

Schnorr TAPS

$\sigma =$



- Improved efficiency and simplicity over generic zero knowledge.
- Verifier derives the challenge directly.

Schnorr TAPS

- *KeyGen*: Generate ATS keypairs for each signer, and keypairs for the combiner (signing) and tracer (encryption).
- *Sign*: Each signer performs an ATS signing operation.
- *Combine*: Perform the ATS combine to generate σ. Encrypt σ to the combiner's public key, then generate a ZKP π that the encryption is correct, authenticated using the tracer's signing key.
- *Verify*: Check the correctness of π relative to the combiner's verification key and the ATS public key.
- *Trace*: Using the tracing decryption key, decrypt σ, then determine the coalition of signers.

Schnorr TAPS

- *KeyGen*: Generate ATS keypairs for each signer, and keypairs for the combiner (signing) and tracer (encryption).
- *Sign*: Each signer performs an ATS signing operation.
- *Combine*: Perform the ATS combine to generate σ. Encrypt σ to the combiner's public key, then generate a ZKP π that the encryption is correct, authenticated using the tracer's signing key.
- *Verify*: Check the correctness of π relative to the combiner's verification key and the ATS public key.
- *Trace*: Using the tracing decryption key, decrypt σ, then determine the coalition of signers.

Schnorr TAPS

- *KeyGen*: Generate ATS keypairs for each signer, and keypairs for the combiner (signing) and tracer (encryption).
- *Sign*: Each signer performs an ATS signing operation.
- *Combine*: Perform the ATS combine to generate σ. Encrypt σ to the combiner's public key, then generate a ZKP π that the encryption is correct, authenticated using the tracer's signing key.
- *Verify*: Check the correctness of π relative to the combiner's verification key and the ATS public key.
- *Trace*: Using the tracing decryption key, decrypt σ, then determine the coalition of signers.

Schnorr TAPS

- *KeyGen*: Generate ATS keypairs for each signer, and keypairs for the combiner (signing) and tracer (encryption).
- *Sign*: Each signer performs an ATS signing operation.
- *Combine*: Perform the ATS combine to generate σ. Encrypt σ to the combiner's public key, then generate a ZKP π that the encryption is correct, authenticated using the tracer's signing key.
- *Verify*: Check the correctness of π relative to the combiner's verification key and the ATS public key.
- *Trace*: Using the tracing decryption key, decrypt σ, then determine the coalition of signers.

Schnorr TAPS

- *KeyGen*: Generate ATS keypairs for each signer, and keypairs for the combiner (signing) and tracer (encryption).
- *Sign*: Each signer performs an ATS signing operation.
- *Combine*: Perform the ATS combine to generate σ. Encrypt σ to the combiner's public key, then generate a ZKP π that the encryption is correct, authenticated using the tracer's signing key.
- *Verify*: Check the correctness of π relative to the combiner's verification key and the ATS public key.
- *Trace*: Using the tracing decryption key, decrypt σ, then determine the coalition of signers.

Schnorr TAPS Relation

$$\mathcal{R}_S = \left\{ (g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1)) \ ; \ (z, \rho, \psi, b_1, \ldots, b_n) \right\}$$

- g, h are generators whose dlog relation is unknown
- (T_0, T_1) is an ElGamal commitment to t
- R is the Schnorr commitment
- c is the Schnorr challenge
- $ct = (c_0, c_1)$ is the encrypted ATS signature z
- ρ, ψ are randomizers
- $b_1, \ldots, b_n$ are bits indicating which signers participated

Schnorr TAPS Relation

$$\mathcal{R}_S = \Big\{(g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1))\ ;\ (z, \rho, \psi, b_1, \ldots, b_n)\Big\}$$

- g, h are generators whose dlog relation is unknown
- (T_0, T_1) is an ElGamal commitment to t
- R is the Schnorr commitment
- c is the Schnorr challenge
- $ct = (c_0, c_1)$ is the encrypted ATS signature z
- ρ, ψ are randomizers
- $b_1, \ldots, b_n$ are bits indicating which signers participated

Schnorr TAPS Relation

$$\mathcal{R}_S = \left\{ (g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1)) \; ; \; (z, \rho, \psi, b_1, \ldots, b_n) \right\}$$

- g, h are generators whose dlog relation is unknown
- (T_0, T_1) is an ElGamal commitment to t
- R is the Schnorr commitment
- c is the Schnorr challenge
- $ct = (c_0, c_1)$ is the encrypted ATS signature z
- ρ, ψ are randomizers
- $b_1, \ldots, b_n$ are bits indicating which signers participated

Schnorr TAPS Relation

$$\mathcal{R}_S = \Big\{(g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1))\ ;\ (z, \rho, \psi, b_1, \ldots, b_n)\Big\}$$

- g, h are generators whose dlog relation is unknown
- (T_0, T_1) is an ElGamal commitment to t
- R is the Schnorr commitment
- c is the Schnorr challenge
- $ct = (c_0, c_1)$ is the encrypted ATS signature z
- ρ, ψ are randomizers
- $b_1, \ldots, b_n$ are bits indicating which signers participated

Schnorr TAPS Relation

$$\mathcal{R}_S = \left\{ (g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1)) \; ; \; (z, \rho, \psi, b_1, \ldots, b_n) \right\}$$

- g, h are generators whose dlog relation is unknown
- (T_0, T_1) is an ElGamal commitment to t
- R is the Schnorr commitment
- c is the Schnorr challenge
- $ct = (c_0, c_1)$ is the encrypted ATS signature z
- ρ, ψ are randomizers
- $b_1, \ldots, b_n$ are bits indicating which signers participated

Schnorr TAPS Relation

$$\mathcal{R}_S = \Big\{(g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1)) \ ; \ (z, \rho, \psi, b_1, \ldots, b_n)\Big\}$$

- g, h are generators whose dlog relation is unknown
- (T_0, T_1) is an ElGamal commitment to t
- R is the Schnorr commitment
- c is the Schnorr challenge
- $ct = (c_0, c_1)$ is the encrypted ATS signature z
- ρ, ψ are randomizers
- $b_1, \ldots, b_n$ are bits indicating which signers participated

Schnorr TAPS Relation

$$\mathcal{R}_S = \Big\{(g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1)) \; ; \; (z, \rho, \psi, b_1, \ldots, b_n)\Big\}$$

- g, h are generators whose dlog relation is unknown
- (T_0, T_1) is an ElGamal commitment to t
- R is the Schnorr commitment
- c is the Schnorr challenge
- $ct = (c_0, c_1)$ is the encrypted ATS signature z
- ρ, ψ are randomizers
- $b_1, \ldots, b_n$ are bits indicating which signers participated

Schnorr TAPS Relation

$$\mathcal{R}_S = \left\{ (g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1)) \; ; \; (z, \rho, \psi, b_1, \ldots, b_n) \right\}$$

- g, h are generators whose dlog relation is unknown
- (T_0, T_1) is an ElGamal commitment to t
- R is the Schnorr commitment
- c is the Schnorr challenge
- $ct = (c_0, c_1)$ is the encrypted ATS signature z
- ρ, ψ are randomizers
- $b_1, \ldots, b_n$ are bits indicating which signers participated

Schnorr TAPS Relation

$$\mathcal{R}_S = \Big\{(g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1))\ ;\ (z, \rho, \psi, b_1, \ldots, b_n)\Big\} \text{ iff}$$

$$(1) \quad g^z = \Big[\prod_{i=1}^{n} (pk_i)^{b_i}\Big]^c \cdot R$$

// Proves z is valid for a subset of public keys in PK

$$(2) \quad c_0 = g^\rho \quad \text{and} \quad c_1 = g^z \cdot pk_t^\rho$$

// Proves the ciphertext is a valid encryption to tracer's public key

$$(3) \quad T_0 = g^\psi \quad \text{and} \quad T_1 = g^{\sum_{i=1}^{n} b_i} \cdot h^\psi$$

// Proves signing quorum contains t signers

$$(4) \quad b_i(1 - b_i) = 0 \qquad \text{for } i = 1, \ldots, n \quad \text{// Proves each } b_i \in \{0, 1\}$$

Schnorr TAPS Relation

$$\mathcal{R}_S = \Big\{ (g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1)) \; ; \; (z, \rho, \psi, b_1, \ldots, b_n) \Big\} \text{ iff}$$

$$(1) \quad g^z = \left[\prod_{i=1}^{n} (pk_i)^{b_i} \right]^c \cdot R$$

// Proves z is valid for a subset of public keys in PK

$$(2) \quad c_0 = g^\rho \quad \text{and} \quad c_1 = g^z \cdot pk_t^\rho$$

// Proves the ciphertext is a valid encryption to tracer's public key

$$(3) \quad T_0 = g^\psi \quad \text{and} \quad T_1 = g^{\sum_{i=1}^{n} b_i} \cdot h^\psi$$

// Proves signing quorum contains t signers

$$(4) \quad b_i(1 - b_i) = 0 \qquad \text{for } i = 1, \ldots, n$$ // Proves each $b_i \in \{0, 1\}$

Schnorr TAPS Relation

$$\mathcal{R}_S = \Big\{(g, h, pk' = (pk_1, \ldots, pk_n), pk_t, T_0, T_1, R, c, ct = (c_0, c_1))\ ;\ (z, \rho, \psi, b_1, \ldots, b_n)\Big\} \text{ iff}$$

$$(1) \quad g^z = \Big[\prod_{i=1}^{n} (pk_i)^{b_i}\Big]^c \cdot R$$

// Proves z is valid for a subset of public keys in PK

$$(2) \quad c_0 = g^\rho \quad \text{and} \quad c_1 = g^z \cdot pk_t^\rho$$

// Proves the ciphertext is a valid encryption to tracer's public key

$$(3) \quad T_0 = g^\psi \quad \text{and} \quad T_1 = g^{\sum_{i=1}^{n} b_i} \cdot h^\psi$$

// Proves signing quorum contains t signers

$$(4) \quad b_i(1 - b_i) = 0 \qquad \text{for } i = 1, \ldots, n \quad \text{// Proves each } b_i \in \{0, 1\}$$

Schnorr TAPS Relation

$$\mathcal{R}_S = \Big\{ (g, h, pk' = (pk_1, \ldots, pk_n), pk_{\mathrm{t}}, T_0, T_1, R, c, ct = (c_0, c_1)) \; ; \; (z, \rho, \psi, b_1, \ldots, b_n) \Big\} \text{ iff}$$

$$(1) \quad g^z = \Big[\prod_{i=1}^{n} (pk_i)^{b_i} \Big]^c \cdot R$$

// Proves z is valid for a subset of public keys in PK

$$(2) \quad c_0 = g^\rho \quad \text{and} \quad c_1 = g^z \cdot pk_{\mathrm{t}}^\rho$$

// Proves the ciphertext is a valid encryption to tracer's public key

$$(3) \quad T_0 = g^\psi \quad \text{and} \quad T_1 = g^{\sum_{i=1}^n b_i} \cdot h^\psi$$

// Proves signing quorum contains t signers

$$(4) \quad b_i(1 - b_i) = 0 \qquad \text{for } i = 1, \ldots, n \quad \text{// Proves each } b_i \in \{0, 1\}$$

Schnorr TAPS Relation

$$\mathcal{R}_S = \Big\{ \big(g, h, pk' = (pk_1, \ldots, pk_n), pk_{\mathrm{t}}, T_0, T_1, R, c, ct = (c_0, c_1)\big) \;;\; \big(z, \rho, \psi, b_1, \ldots, b_n\big) \Big\} \text{ iff}$$

$$(1) \quad g^z = \Big[\prod_{i=1}^{n} (pk_i)^{b_i} \Big]^c \cdot R$$

// Proves z is valid for a subset of public keys in PK

$$(2) \quad c_0 = g^\rho \quad \text{and} \quad c_1 = g^z \cdot pk_{\mathrm{t}}^\rho$$

// Proves the ciphertext is a valid encryption to tracer's public key

$$(3) \quad T_0 = g^\psi \quad \text{and} \quad T_1 = g^{\sum_{i=1}^{n} b_i} \cdot h^\psi$$

// Proves signing quorum contains t signers

$$(4) \quad b_i(1 - b_i) = 0 \qquad \text{for } i = 1, \ldots, n \quad \text{// Proves each } b_i \in \{0, 1\}$$

Performance of Schnorr TAPS

	Public Key Size		Signature Size	
	$\mathbb{G}$	$\mathbb{Z}_q$	$\mathbb{G}$	$\mathbb{Z}_q$
Sigma	$\approx 2n$	0	$\approx n$	$\approx 2n$
Bulletproofs	$\approx n + \frac{n}{40}$	0	$\approx \frac{n}{40}$	4

- Both constructions reduce to discrete logarithm assumptions.

Takeaways

- TAPS are a new type of threshold signature with both privacy and accountability.
- We define a generic construction employing an encrypted ATS.
- We define a Schnorr construction that leverages the structure of Schnorr to simplify the zero-knowledge statement.
- We define sigma and bulletproofs instantiations of the zero-knowledge argument.

Thank You!

Takeaways

- TAPS are a new type of threshold signature with both privacy and accountability.
- We define a generic construction employing an encrypted ATS.
- We define a Schnorr construction that leverages the structure of Schnorr to simplify the zero-knowledge statement.
- We define sigma and bulletproofs instantiations of the zero-knowledge argument.

Thank You!

Takeaways

- TAPS are a new type of threshold signature with both privacy and accountability.
- We define a generic construction employing an encrypted ATS.
- We define a Schnorr construction that leverages the structure of Schnorr to simplify the zero-knowledge statement.
- We define sigma and bulletproofs instantiations of the zero-knowledge argument.

Thank You!

Takeaways

- TAPS are a new type of threshold signature with both privacy and accountability.
- We define a generic construction employing an encrypted ATS.
- We define a Schnorr construction that leverages the structure of Schnorr to simplify the zero-knowledge statement.
- We define sigma and bulletproofs instantiations of the zero-knowledge argument.

Thank You!

Takeaways

- TAPS are a new type of threshold signature with both privacy and accountability.
- We define a generic construction employing an encrypted ATS.
- We define a Schnorr construction that leverages the structure of Schnorr to simplify the zero-knowledge statement.
- We define sigma and bulletproofs instantiations of the zero-knowledge argument.

Thank You!