DKLs: Threshold ECDSA

In Three Rounds, From VOLE and Commitments

Jack Doerner, NIST TCPT3, October 7 2026

Who We Are

An academic + industrial team, including the original DKLs protocol authors, plus authors of the major sub-protocols included in the package.

More Information:
http://dkls.info

Jack Doerner
U of Virginia



Yashvanth Kondi
Silence Labs



Eysa Lee
Barnard College





abhi shelat
Northeastern U
+ Google



Peter Rindal
Category Labs



Lance Roy
IBM Research

What Must We Securely Compute?

DLKeyGen($\mathbb{G}, G, q$) :

$\mathsf{sk} \leftarrow \mathbb{Z}_q$

$\mathsf{pk} = \mathsf{sk} \cdot G$

output $(\mathsf{sk}, \mathsf{pk})$

Prime-order group.
Additive notation.

ECDSASign(sk, m) :

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

$e = H(m)$

Similar to
KG and Schnorr

SHA256

What Must We Securely Compute?

$\mathrm{DLKeyGen}(\mathbb{G}, G, q)$:

$\mathsf{sk} \leftarrow \mathbb{Z}_q$

$\mathsf{pk} = \mathsf{sk} \cdot G$

output $(\mathsf{sk}, \mathsf{pk})$

$\mathrm{ECDSASign}(\mathsf{sk}, m)$:

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

$e = H(m)$

$$s = \frac{e + \mathsf{sk} \cdot r_x}{k}$$

multiplication of secret values

output $\sigma = (s, R)$

What Must We Securely Compute?

$\mathrm{DLKeyGen}(\mathbb{G}, G, q)$:

$\mathsf{sk} \leftarrow \mathbb{Z}_q$

$\mathsf{pk} = \mathsf{sk} \cdot G$

output $(\mathsf{sk}, \mathsf{pk})$

$\mathrm{ECDSASign}(\mathsf{sk}, m)$:

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

$e = H(m)$

$$s = \frac{e + \mathsf{sk} \cdot r_x}{k}$$

output $\sigma = (s, R)$

multiplication of secret values

modular inverse

What Must We Securely Compute?

$\mathrm{DLKeyGen}(\mathbb{G}, G, q)$:

$\mathsf{sk} \leftarrow \mathbb{Z}_q$

$\mathsf{pk} = \mathsf{sk} \cdot G$

output $(\mathsf{sk}, \mathsf{pk})$

$\mathrm{ECDSASign}(\mathsf{sk}, m)$:

$k \leftarrow \mathbb{Z}_q$

$R = k \cdot G$

$e = H(m)$

$$s = \frac{e + \mathsf{sk} \cdot r_x}{k}$$

output $\sigma = (s, R)$

multiplication of secret values

modular inverse

We can view the computation of s as a simple circuit over the field $\mathbb{Z}_q$.

We have ~40 years of literature on secure multiparty computation over such fields.

What Makes ECDSA Special

Challenges:

1. Most techniques focus on basic field operations. Inverse is nontrivial.
2. Verifying honest behavior involves a nonlinear relation about a mix of $\mathbb{Z}_q$ and $\mathbb{G}$ elements.
3. Circuit is too small for amortization to kick in, but applications are highly performance-sensitive.

$\mathrm{ECDSASign}(\mathsf{sk}, m)$:

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

$$R = k \cdot G$$

$$e = H(m)$$

$$s = \frac{e + \mathsf{sk} \cdot r_x}{k}$$

multiplication of secret values

modular inverse

$$\text{output } \sigma = (s, R)$$

We can view the computation of s as a simple circuit over the field $\mathbb{Z}_q$.

We have ~40 years of literature on secure multiparty computation over such fields.

What Makes ECDSA Special

Challenges:

1. Most techniques focus on basic field operations. Inverse is nontrivial.
2. Verifying honest behavior involves a nonlinear relation about a mix of $\mathbb{Z}_q$ and $\mathbb{G}$ elements.
3. Circuit is too small for amortization to kick in, but applications are highly performance-sensitive.

Prototypical Threshold ECDSA Paper:

1. Rewrite ECDSA in terms of "friendly" operations (usually $+$ and $\times$).
2. Consistency machinery relating secrets in $\mathbb{Z}_q$ to both a public product and public values in $\mathbb{G}$.
3. Efficient (usually "one-shot") protocol technique for $\times$.

We can view the computation of s as a simple circuit over the field $\mathbb{Z}_q$.

We have ~40 years of literature on secure multiparty computation over such fields.

Challenges:

1. Most techniques focus on basic field operations. Inverse is nontrivial.
2. Verifying honest behavior involves a nonlinear relation about a mix of $\mathbb{Z}_q$ and $\mathbb{G}$ elements.
3. Circuit is too small for amortization to kick in, but applications are highly performance-sensitive.

Prototypical Threshold ECDSA Paper:

1. Rewrite ECDSA in terms of "friendly" operations (usually + and ×).
2. Consistency machinery relating secrets in $\mathbb{Z}_q$ to both a public product and public values in $\mathbb{G}$.
3. Efficient (usually "one-shot") protocol technique for ×.

$$s = \frac{e + \mathsf{SK} \cdot r_x}{}$$

Tradeoffs (efficiency, assumptions, security model) induced by the structure of ECDSA seem much more pronounced than for Schnorr or RSA signing.

This has lead to dozens of papers, very distinct approaches.

There is no one-size-fits-all solution.

Nevertheless, a standard defines one solution.

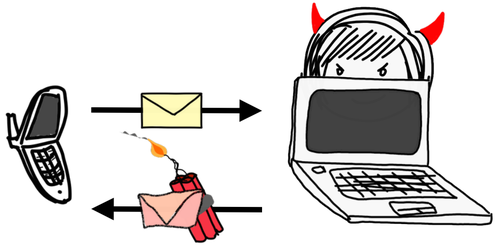
Where Do We Fit?

- Corruption Type



Static, Dishonest Majority

- Adversarial Behavior



Malicious

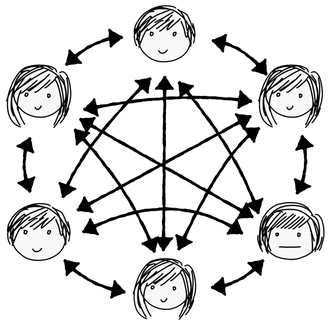
Where Do We Fit?

- Corruption Type



Static, Dishonest Majority

- Network

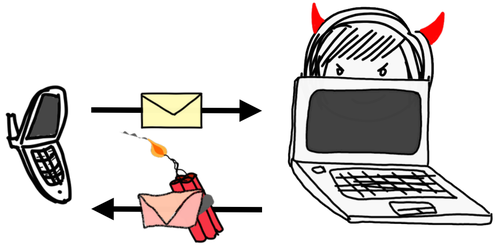


Async, Point-to-Point

$\Longrightarrow$ Minimal requirements on deployment environment

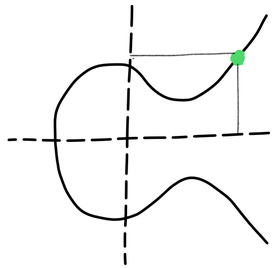
$\Longrightarrow$ No need for broadcast protocols or clocks

- Adversarial Behavior



Malicious

- Assumptions



ECDH, ROM

$\Longrightarrow$ Minimal exposure to advances in cryptanalysis

$\Longrightarrow$ Primitives are already NIST-standardized

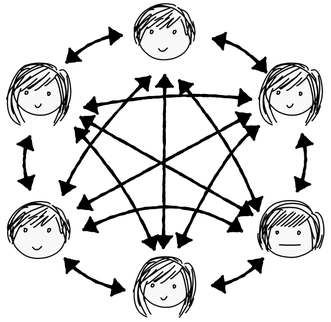
Where Do We Fit?

- Corruption Type



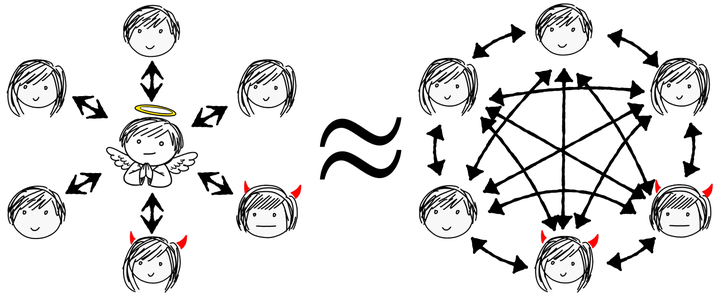
Static, Dishonest Majority

- Network



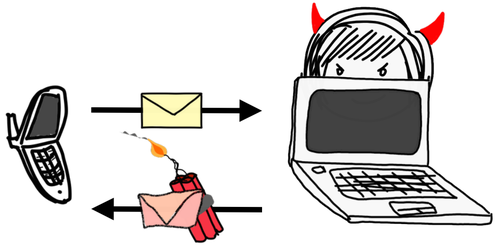
Async, Point-to-Point

- Security Model



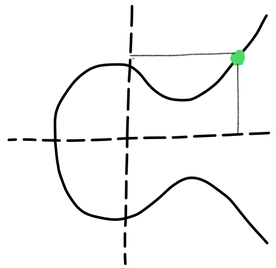
Universally Composable

- Adversarial Behavior



Malicious

- Assumptions



ECDH, ROM

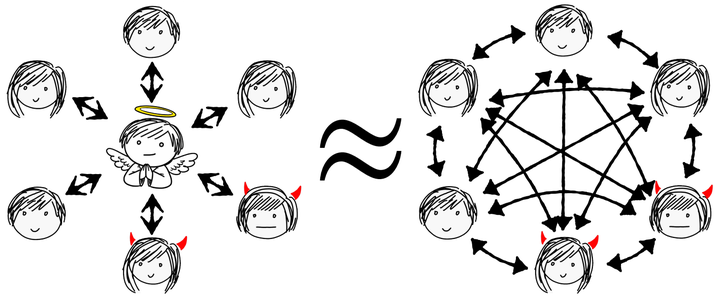
- Ideal Adversary's Power



Abort, *No* Nonce Bias

Where Do We Fit?

- Security Model



Universally Composable

Safe to use anywhere a trusted signer is, without extra analysis ⟸

- Ideal Adversary's Power



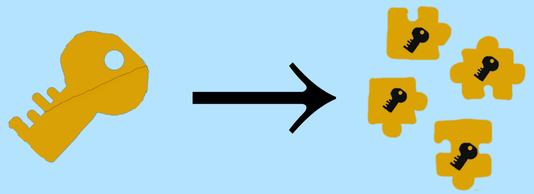
Abort, *No* Nonce Bias

Appropriate for "permissioned" contexts ⟸

In this talk

- Intro and Where We Fit
- A Brief Intro to DKLs
    1. Rewriting ECDSA
    2. Consistency Machinery
    3. Secure Multiplication
- Implementations and Performance
- What's in our Submission

1. Rewriting the ECDSA Equation



Linear Secret Sharing

Party i knows share k_i

$$\text{ECDSASign}([\mathsf{sk}], m):$$

$$[k] \leftarrow \mathbb{Z}_q$$

$$R = [k] \cdot G$$

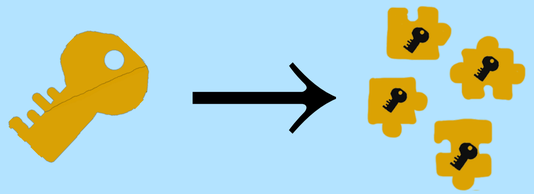
$$e = H(m)$$

$$s = \frac{e + r_x \cdot [\mathsf{sk}]}{[k]}$$

$$\sigma = (s, r_x)$$

output σ

1. Rewriting the ECDSA Equation



Linear Secret Sharing

Party i knows share k_i

$\mathrm{ECDSASign}([\mathsf{sk}], m)$:

$[k] \leftarrow \mathbb{Z}_q$

$R = \mathsf{Reveal}\ [k] \cdot G$

$e = H(m)$

$s = \mathsf{Reveal}\ \dfrac{e + r_x \cdot [\mathsf{sk}]}{[k]}$

$\sigma = (s, r_x)$

output σ

1. Rewriting the ECDSA Equation

$$\begin{aligned}
&\text{ECDSASign}([\mathsf{sk}], m):\\
&\qquad [k] \leftarrow \mathbb{Z}_q\,, [\phi] \leftarrow \mathbb{Z}_q\\
&\qquad R = \mathsf{Reveal}\,[k]\cdot G\\
&\qquad e = H(m)\\
&\qquad s = \mathsf{Reveal}\ \frac{e + r_x \cdot [\mathsf{sk}]}{[k]} \cdot \frac{[\phi]}{[\phi]}\\
&\qquad \sigma = (s, r_x)\\
&\text{output } \sigma
\end{aligned}$$

[Bar-Ilan Beaver 89]

1. Rewriting the ECDSA Equation

$$\begin{aligned}
&\text{ECDSASign}([\mathsf{sk}], m):\\
&\quad [k] \leftarrow \mathbb{Z}_q, [\phi] \leftarrow \mathbb{Z}_q\\
&\quad R = \mathsf{Reveal}\ [k] \cdot G\\
&\quad e = H(m)\\
&\quad s = \mathsf{Reveal}\ \frac{e \cdot [\phi] + r_x \cdot [\mathsf{sk} \cdot \phi]}{[k \cdot \phi]}\\
&\quad \sigma = (s, r_x)\\
&\text{output } \sigma
\end{aligned}$$

Call this value α
Fixed by β and s

Uniform because ϕ masks k
Call this value β; reveal it

1. Rewriting the ECDSA Equation

$$\begin{aligned}
&\mathrm{ECDSASign}([\mathsf{sk}], m):\\
&\quad [k] \leftarrow \mathbb{Z}_q, [\phi] \leftarrow \mathbb{Z}_q\\
&\quad R = \mathsf{Reveal}\ [k] \cdot G\\
&\quad e = H(m)\\
&\quad \alpha = \mathsf{Reveal}\ e \cdot [\phi] + r_x \cdot [\mathsf{sk} \cdot \phi]\\
&\quad \beta = \mathsf{Reveal}\ [k \cdot \phi]\\
&\quad s = \alpha/\beta\\
&\quad \sigma = (s, r_x)\\
&\text{output } \sigma
\end{aligned}$$

Independent of everything else

Non-interactive once we have $[\mathsf{sk} \cdot \phi]$

Two multiplications of secrets, independently

Non-interactive

3-Round Protocol for Honest Parties

Input $[\mathsf{sk}], \mathsf{PK}$

Local Sample $[k]$ $[\phi]$

Establish $R = [k] \cdot G$ (without bias)

Round 1: Every Party i: Commit $\left(R_i = k_i \cdot G\right)$

Round 2: Every Party i: Release R_i, set $R = \Sigma_i \; R_i$

3-Round Protocol for Honest Parties

Input $[\mathsf{sk}], \mathsf{PK}$

Local Sample $[k]$ $[\phi]$

Round 1 / Round 2

Establish $R = [k] \cdot G$ (without bias)

Sec Mult $[k \cdot \phi]$

Sec Mult $[\mathsf{sk} \cdot \phi]$

Local $[\alpha] = e \cdot [\phi] + r_x \cdot [\mathsf{sk} \cdot \phi], \quad [\beta] = [k \cdot \phi]$

Round 3

Reveal (α, β)

Output $\sigma = (\alpha/\beta, r_x)$

2. Consistency Machinery

Input [sk], PK

Local

Round 1

Round 2

Local

Round 3

Reveal (α, β)

Output $\sigma = (\alpha/\beta, r_x)$

Consistency machinery hardens the protocol against malicious behavior.

To develop intuition:
What could go wrong?

(the full analysis is simulation-based)

2. Consistency Machinery

Input $[\mathsf{sk}], \mathsf{PK}$

Local Sample $[k]$ $[\phi]$

Round 1 / Round 2

Establish $R = [k] \cdot G$ (without bias)

Sec Mult $[k \cdot \phi]$

Sec Mult $[\mathsf{sk} \cdot \phi]$

Local $[\alpha] = e \cdot [\phi] + r_x \cdot [\mathsf{sk} \cdot \phi], \quad [\beta] = [k \cdot \phi]$

Round 3

Reveal (α, β)

Send different values of (α_i, β_i, R_i) to different people.

Output $\sigma = (\alpha/\beta, r_x)$

Lightweight solution: verify* σ, echo commitments

*including the y-coordinate of R [Segev 26]

2. Consistency Machinery

Local — Input $[\mathsf{sk}], \mathsf{PK}$

Local — Sample $[k]$ $[\phi]$

Cheating here is an independent problem *if* we can treat these as ideal objects

Round 1 / Round 2 — Establish $R = [k] \cdot G$ (without bias)

Sec Mult $[k \cdot \phi]$

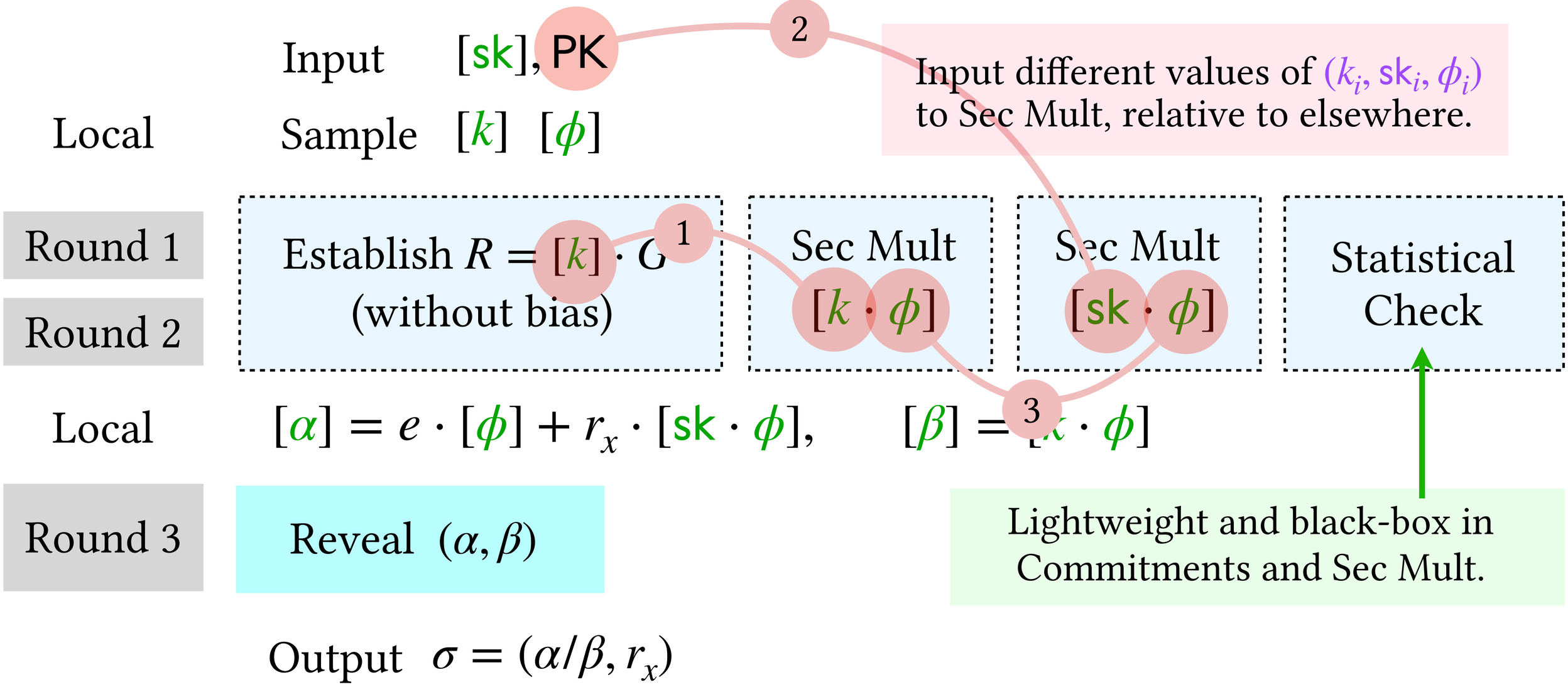
Sec Mult $[\mathsf{sk} \cdot \phi]$

Local — $[\alpha] = e \cdot [\phi] + r_x \cdot [\mathsf{sk} \cdot \phi], \qquad [\beta] = [k \cdot \phi]$

Round 3 — Reveal (α, β)

Output $\sigma = (\alpha/\beta, r_x)$

2. Consistency Machinery





$$[k \cdot \phi] = \left[\sum_{i,j \in [n]} k_i \cdot \phi_j \right]$$

- Party i can compute $k_i \cdot \phi_i$

[sk]
[k]
[ϕ]

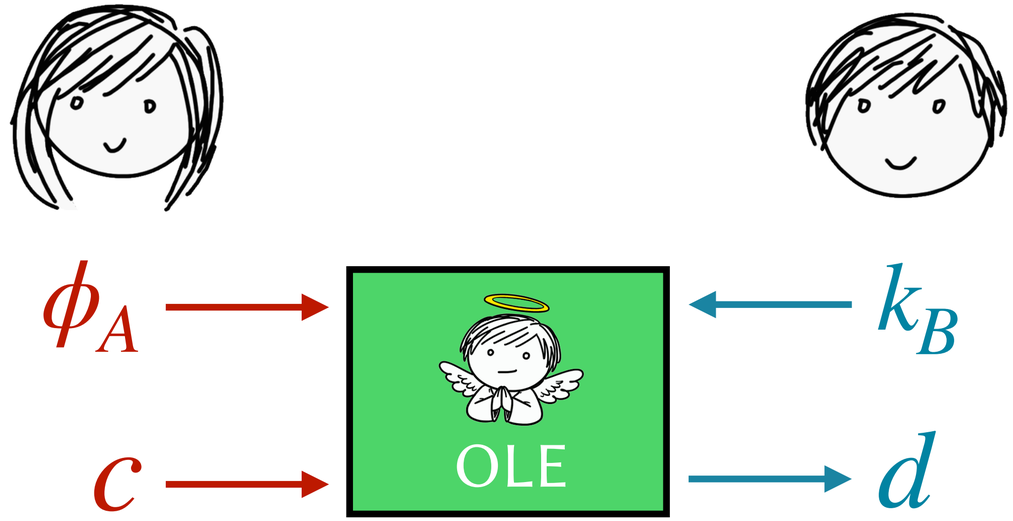
Sec Mult
[$k \cdot \phi$]

Sec Mult
[sk $\cdot \phi$]

Statistical Check

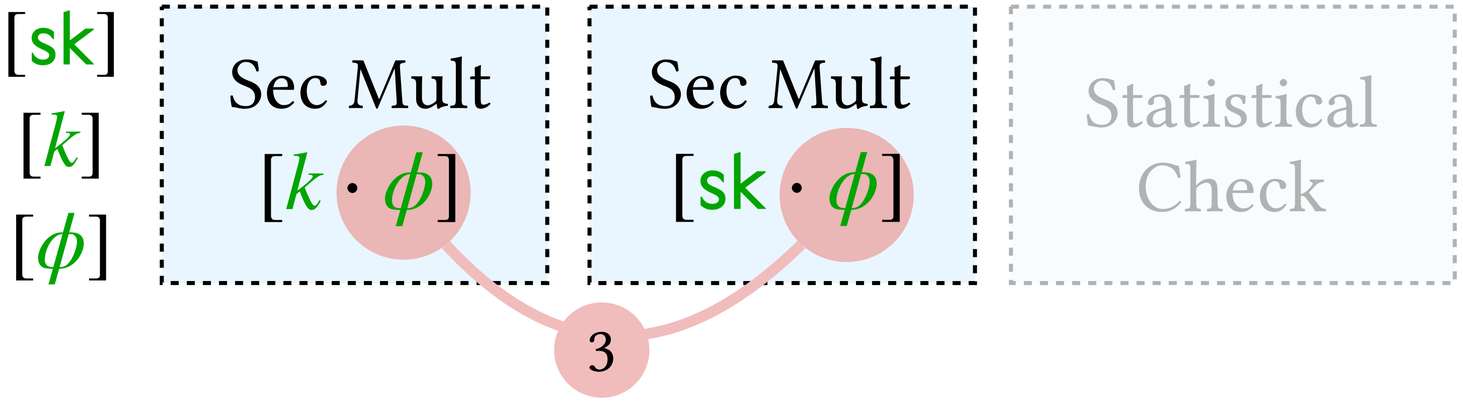
$$[k \cdot \phi] = \left[\sum_{i,j \in [n]} k_i \cdot \phi_j \right]$$

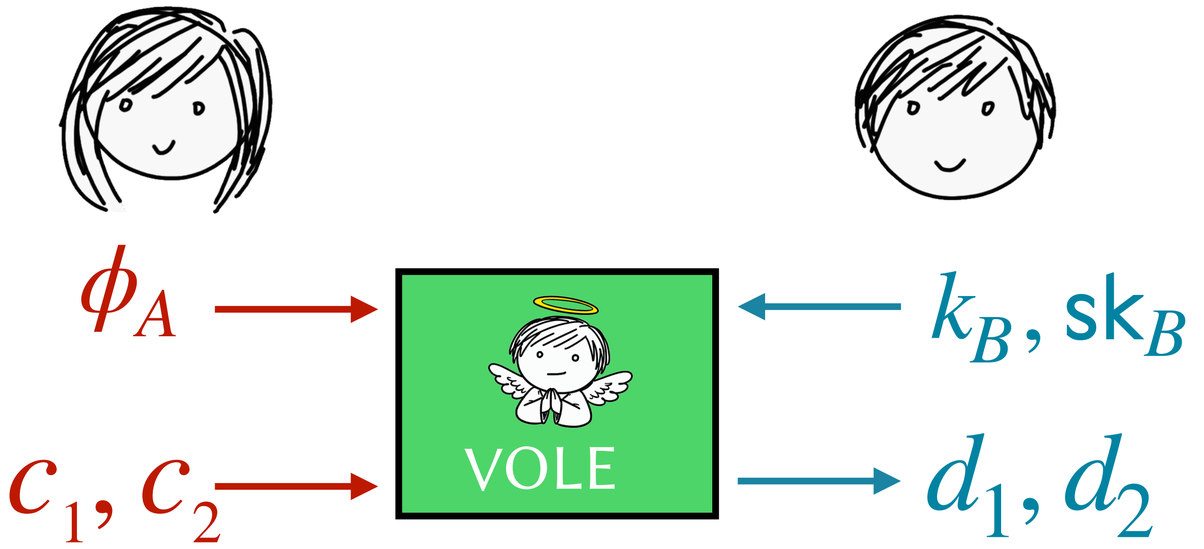
- Party i can compute $k_i \cdot \phi_i$
- To compute $[k_i \cdot \phi_j] : j \neq i$
  A 2-party multiplication protocol.



$$c + d = \phi_A \cdot k_B$$

If  chooses a random c,
then d info-theoretically hides a from 





$$c_1 + d_1 = \phi_A \cdot k_B$$
$$c_2 + d_2 = \phi_A \cdot \mathsf{sk}_B$$

Consistency for Free

[sk]
[k]
[ϕ]

Sec Vector Mult

[$k \cdot \phi$] [sk $\cdot \phi$]

3

Statistical Check



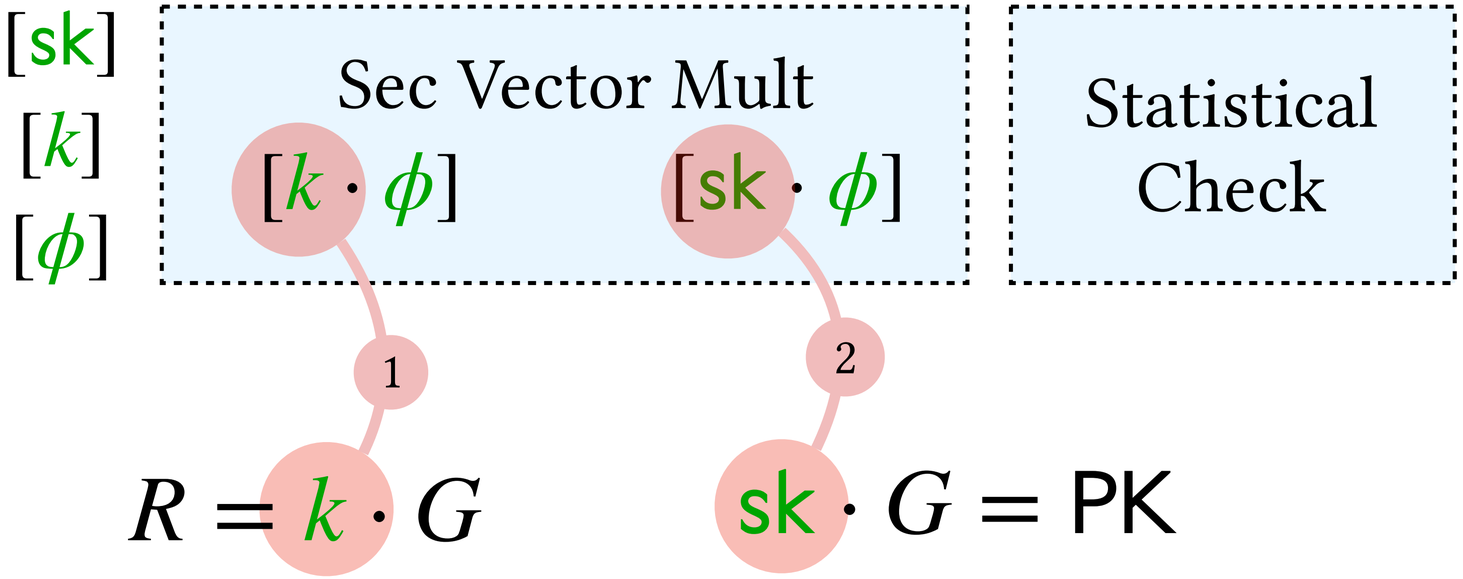


ϕ_A → VOLE ← k_B, sk_B

c_1, c_2 → VOLE → d_1, d_2

$$c_1 + d_1 = \phi_A \cdot k_B$$
$$c_2 + d_2 = \phi_A \cdot \mathsf{sk}_B$$

Consistency for Free

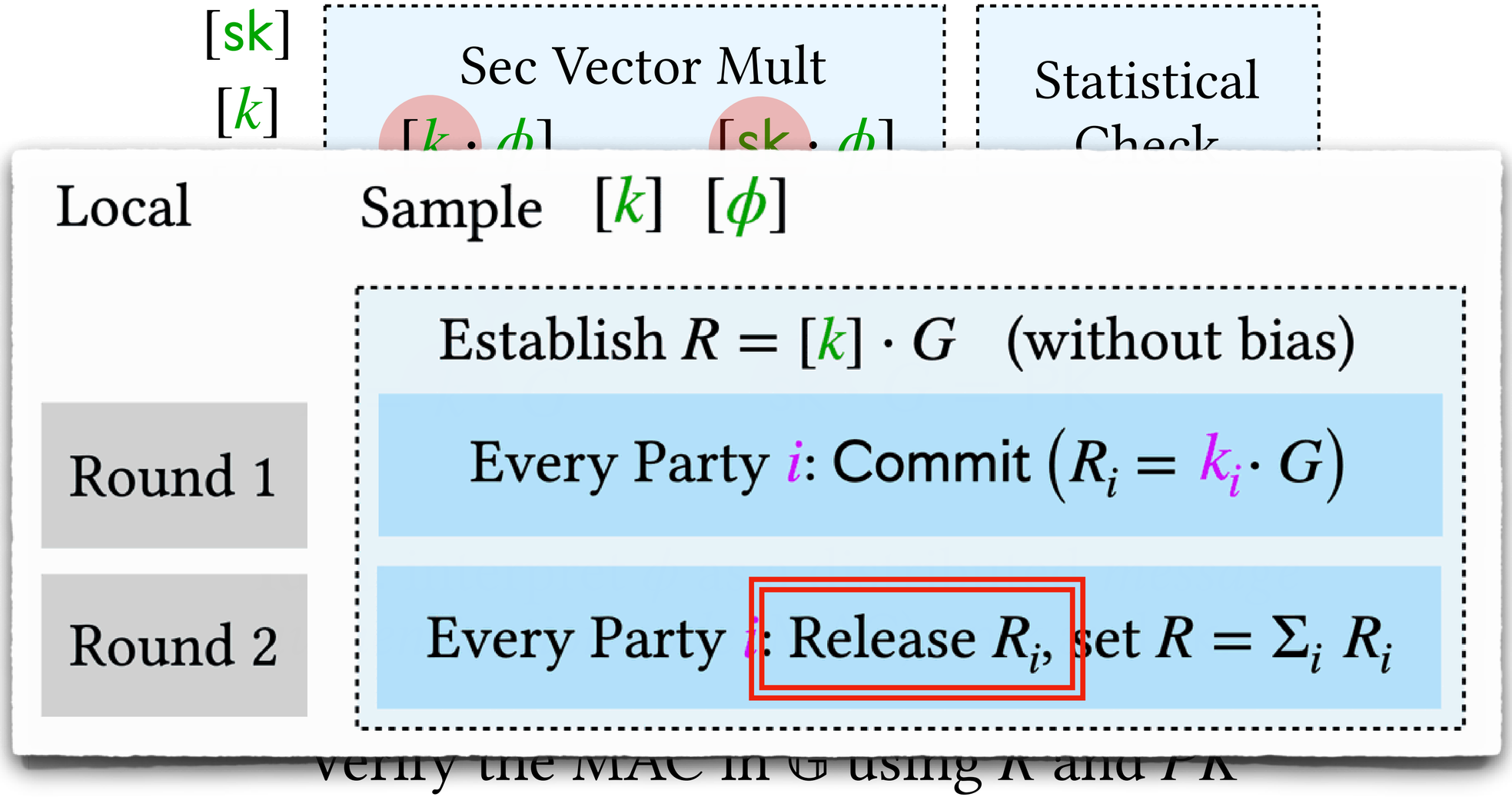


Idea: interpret ϕ as a distributed *message authentication code* (MAC) on k and sk.

Verify the MAC in $\mathbb{G}$ using R and PK

Key Fact: for every Party i, $R_i = k_i \cdot G$ is public

(we also learned $\mathsf{PK}_i = \mathsf{sk}_i \cdot G$ during Distributed Key Generation)



Key Fact: for every Party i, $R_i = k_i \cdot G$ is public

(we also learned $\mathsf{PK}_i = \mathsf{sk}_i \cdot G$ during Distributed Key Generation)

Idea: interpret ϕ as a distributed *message authentication code* (MAC) on k and sk.

Key Fact:
Everyone Knows
$R_B = k_B \cdot G$





Claim:
$k_B \cdot G = R_B$

$\phi_A \rightarrow$ OLE $\leftarrow k_B$

$c \rightarrow$ OLE $\rightarrow d$

$T = \phi_A \cdot R_B - c \cdot G$

$D = d \cdot G$

$c + d = a \cdot b$

D ←

$T \stackrel{?}{=} D$

If they match, the **claim** is convincing

Idea: interpret ϕ as a distributed *message authentication code* (MAC) on k and sk.

Key Fact:
Everyone Knows
$R_B = k_B \cdot G$





Claim:
$k_B \cdot G = R_B$

ϕ_A —

c —

c

G

$T = \phi_A \cdot R_B - c \cdot G$

$T \stackrel{?}{=} D$

If they match, the **claim** is convincing

For Security We Need:

Privacy: no new information is leaked.

Soundness: malicious  cannot convince  of a false claim, except with negligible probability.

Idea: interpret ϕ as a distributed *message authentication code* (MAC) on k and sk.

Key Fact:
Everyone Knows
$R_B = k_B \cdot G$





Claim:
$k_B \cdot G = R_B$

ϕ_A → | ← k_B



c → | → d

$T = \phi_A \cdot R_B - c \cdot G$

$D = d \cdot G$

$c + d = a \cdot b$

D ←

$T \stackrel{?}{=} D$

If they match, the **claim** is convincing

Idea: interpret ϕ as a distributed *message authentication code* (MAC) on k and sk.

Key Fact:
Everyone Knows
$R_B^* = R_B + \Delta$





Claim:
$k_B \cdot G = R_B$



$T = \phi_A \cdot R_B - c \cdot G$

$D = d \cdot G$

$c + d = a \cdot b$

D^*

$T \stackrel{?}{=} D^*$

If they match, the **claim** is convincing

Idea: interpret ϕ as a distributed *message authentication code* (MAC) on k and sk.

Key Fact:
Everyone Knows
$R_B^* = R_B + \Delta$



$\phi_A \longrightarrow$

$c \longrightarrow$

$c +$

$T = \phi_A \cdot R_B - c \cdot G$

$\longleftarrow$

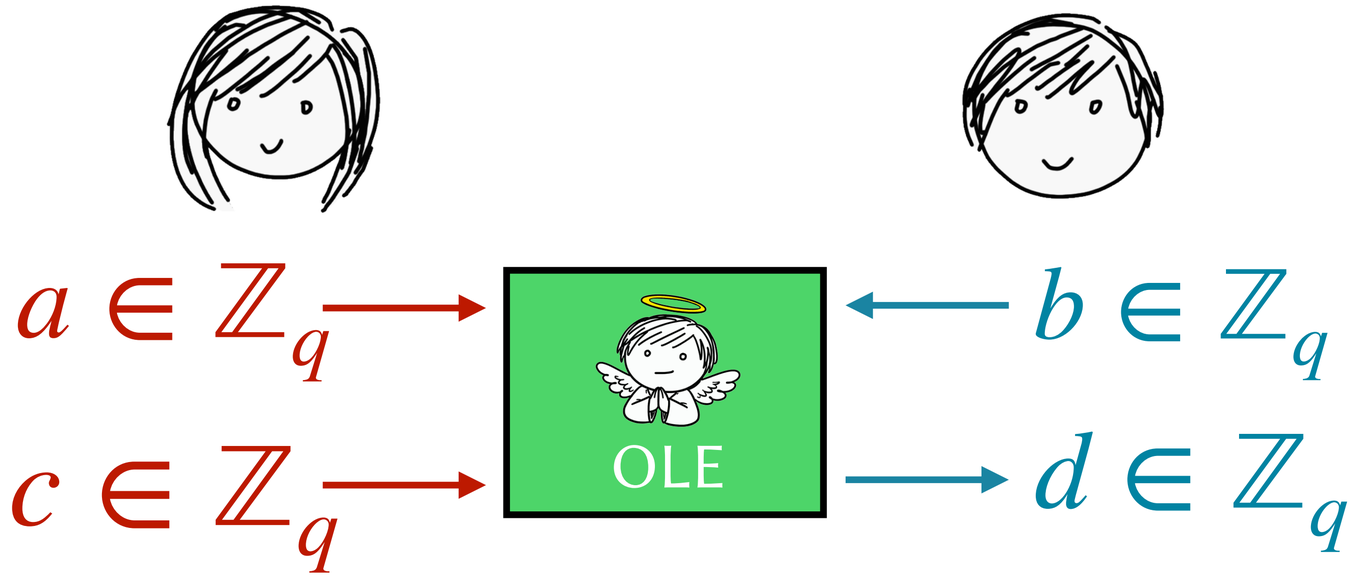
$T \stackrel{?}{=} D^*$

If they match, the **claim** is convincing

If  is *Dishonest*

1. $D = \left(\phi_A \cdot k_B - c\right) \cdot G$
2. $T = \phi_A \cdot R_B^* - c \cdot G$
   $= \phi_A \cdot \Delta + \phi_A \cdot k_B \cdot G - c \cdot G$
3.  must send $D^* = D + \phi_A \cdot \Delta$
4. ϕ_A is info-theoretically hidden from 
5.  succeeds with probability $1/q$ which is negligible

3. Secure Multiplication (i.e. Instantiating OLE)



Instantiating OLE





$a \in \mathbb{Z}_q \rightarrow$

$c \in \mathbb{Z}_q \rightarrow$





$\leftarrow b \in \mathbb{Z}_q$

random in outer protocol
when parties behave honestly

We can sample them here
and make them *outputs*

$\rightarrow d \in \mathbb{Z}_q$

$$d = a \cdot b - c$$

The Core Idea (Semi-Honest): A Two-Party Randomness Extractor, Linear in Seed





$a \in \mathbb{Z}_q \rightarrow$



$\vec{h}$



$\vec{y} \leftarrow \mathbb{Z}_q^\ell$

$c = \langle \vec{h}, \vec{y} \rangle$

$\vec{h} \leftarrow \mathbb{Z}_q^\ell$

$\vec{x} \leftarrow \{0,1\}^\ell$

$b = \langle \vec{h}, \vec{x} \rangle$

$c \in \mathbb{Z}_q \leftarrow$

Inner Product Extractor
[ILL89]

$\rightarrow b \in \mathbb{Z}_q$

$\rightarrow d \in \mathbb{Z}_q$

$$d = a \cdot b - c$$

The Core Idea (Semi-Honest): A Two-Party Randomness Extractor, Linear in Seed





$a \in \mathbb{Z}_q \rightarrow$



$\vec{h}$



$\vec{y} \leftarrow \mathbb{Z}_q^\ell$

$c = \langle \vec{h}, \vec{y} \rangle$

What they must securely compute:

$$d = a \cdot b - c$$
$$= \langle a \cdot \vec{h}, \vec{x} \rangle - \langle \vec{h}, \vec{y} \rangle$$
$$= \sum_i (a \cdot x_i - y_i) \cdot h_i$$

IT-hides a from 

OK to reveal to 

$\vec{h} \leftarrow \mathbb{Z}_q^\ell$

$\vec{x} \leftarrow \{0,1\}^\ell$

$b = \langle \vec{h}, \vec{x} \rangle$

$c \in \mathbb{Z}_q \leftarrow$

$\rightarrow b \in \mathbb{Z}_q$

$\rightarrow d \in \mathbb{Z}_q$

$$d = a \cdot b - c$$

The Core Idea (Semi-Honest): A Two-Party Randomness Extractor, Linear in Seed





$a \in \mathbb{Z}_q \rightarrow$

$c \in \mathbb{Z}_q \leftarrow$



$\vec{h}$



$\vec{y} \leftarrow \mathbb{Z}_q^\ell$

$c = \langle \vec{h}, \vec{y} \rangle$

What they must securely compute:

$\delta_i = a \cdot x_i - y_i$

ℓ OLE instances.
Each input of  is **1 bit**.

$\vec{h} \leftarrow \mathbb{Z}_q^\ell$

$\vec{x} \leftarrow \{0,1\}^\ell$

$b = \langle \vec{h}, \vec{x} \rangle$

$d = \langle \vec{h}, \vec{\delta} \rangle$

$\rightarrow b \in \mathbb{Z}_q$

$\rightarrow d \in \mathbb{Z}_q$

$$d = a \cdot b - c$$

The Core Idea (Semi-Honest): A Two-Party Randomness Extractor, Linear in Seed





$a \in \mathbb{Z}_q \rightarrow$

$c \in \mathbb{Z}_q \leftarrow$

$\rightarrow b \in \mathbb{Z}_q$

$\rightarrow d \in \mathbb{Z}_q$

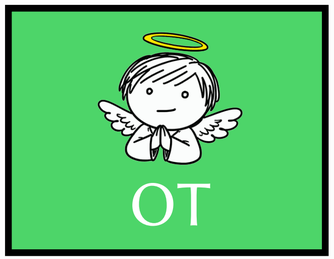
Oblivious Transfer





$m_0 \in \{0,1\}^* \rightarrow$

$m_1 \in \{0,1\}^* \rightarrow$



$\leftarrow x \in \{0,1\}$

$\rightarrow m_x$

The Fundamental Primitive of MPC

$$d = a \cdot b - c$$

The Core Idea (Semi-Honest): A Two-Party Randomness Extractor, Linear in Seed





$a \in \mathbb{Z}_q \rightarrow$



$\vec{h}$



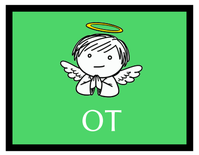
$\vec{y} \leftarrow \mathbb{Z}_q^\ell$

$c = \langle \vec{h}, \vec{y} \rangle$

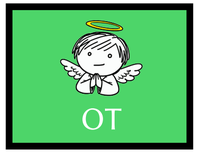
$\vec{h} \leftarrow \mathbb{Z}_q^\ell$

$\vec{x} \leftarrow \{0,1\}^\ell$

$b = \langle \vec{h}, \vec{x} \rangle$

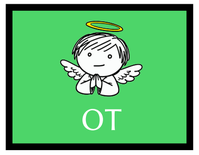
$-y_1 \rightarrow$  $\leftarrow x_1$

$a - y_1 \rightarrow$ OT $\rightarrow \delta_1$

$-y_2 \rightarrow$  $\leftarrow x_2$

$a - y_2 \rightarrow$ OT $\rightarrow \delta_2$

$\vdots$

$-y_\ell \rightarrow$  $\leftarrow x_\ell$

$a - y_\ell \rightarrow$ OT $\rightarrow \delta_\ell$

$\delta_i = a \cdot x_i - y_i$

$d = \langle \vec{h}, \vec{\delta} \rangle$

$c \in \mathbb{Z}_q \leftarrow$

$\rightarrow b \in \mathbb{Z}_q$

$\rightarrow d \in \mathbb{Z}_q$

$$d = a \cdot b - c$$

Malicious Security





$a \in \mathbb{Z}_q \rightarrow$

$\vec{h}$



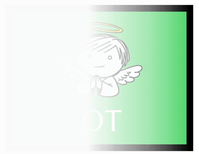
$\vec{h} \leftarrow \mathbb{Z}_q^\ell$

$\vec{x} \leftarrow \{0,1\}^\ell$

$b = \langle \vec{h}, \vec{x} \rangle$

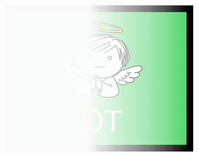
A corrupt  can *only* cheat by sampling $\vec{x}, \vec{h}$ from the wrong distributions.

Attacks of this kind are equivalent to choosing a non-uniform b, which is harmless.



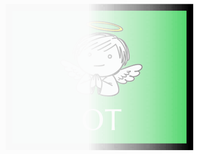
$\leftarrow x_1$

$\rightarrow \delta_1$



$\leftarrow x_2$

$\rightarrow \delta_2$



$\leftarrow x_\ell$

$\rightarrow \delta_\ell$

$d = \langle \vec{h}, \vec{\delta} \rangle$

$c \in \mathbb{Z}_q \leftarrow$

$\rightarrow b \in \mathbb{Z}_q$

$\rightarrow d \in \mathbb{Z}_q$

$$d = a \cdot b - c$$

Malicious Security





$a \in \mathbb{Z}_q \rightarrow$



$\vec{y} \leftarrow \mathbb{Z}_q^\ell$

$c = \langle \vec{h}, \vec{y} \rangle$

$\leftarrow \vec{h}$

$-y_1 \rightarrow$ OT

$a - y_1 \rightarrow$

$-y_2 \rightarrow$ OT

$a - y_2 \rightarrow$

$-y_\ell \rightarrow$ OT

$a - y_\ell \rightarrow$

A corrupt  can cheat by using inconsistent $a, \vec{y}$ values in inputs to different OTs.

To prevent this, we can add another *statistical check* that catches inconsistencies with overwhelming probability.

Problem:  can only run the statistical check the message δ_i that he actually receives.

Cheating on *one* message in OT i amounts to guessing x_i.

$c \in \mathbb{Z}_q \leftarrow$

$\rightarrow b \in \mathbb{Z}_q$

$\rightarrow d \in \mathbb{Z}_q$

$d = a \cdot b - c$

Malicious Security





$a \in \mathbb{Z}_q \rightarrow$



$\vec{y} \leftarrow \mathbb{Z}_q^\ell$

$c = \langle \vec{h}, \vec{y} \rangle$

$-y_1 + \Delta_1$

$a - y_1$

$-y_2$

$a - y_2 + \Delta_2$

$-y_\ell + \Delta_\ell$

$a - y_\ell$

A corrupt  can cheat by using inconsistent $a, \vec{y}$ values in inputs to different OTs.

To prevent this, we can add another *statistical check* that catches inconsistencies with overwhelming probability.

Problem:  can only run the statistical check the message δ_i that he actually receives.

Cheating on *one* message in OT i amounts to guessing x_i.

If the check passes,  learns her guesses were right.

$c \in \mathbb{Z}_q \leftarrow$

$\rightarrow b \in \mathbb{Z}_q$

$\rightarrow d \in \mathbb{Z}_q$

$$d = a \cdot b - c$$

Malicious Security





$a \in \mathbb{Z}_q \rightarrow$



$\vec{h}$

$\vec{y} \leftarrow \mathbb{Z}_q^\ell$

$c = \langle \vec{h}, \vec{y} \rangle$

$-y_1 + \Delta_1$

$a - y_1$

$-y_2$

$a - y_2 + \Delta_2$

$-y_\ell + \Delta_\ell$

$a - y_\ell$

Cheating on *one* message in OT i amounts to guessing x_i.

If the check passes,  learns her guesses were right.

We show that this attack can be interpreted as adversarial bias in the entropy source $\vec{x}$ of the extractor $b = \langle \vec{h}, \vec{x} \rangle$.

A new analysis of the inner product extractor enables us to find an ℓ for which this bias is harmless [Doerner Roy 26].

$c \in \mathbb{Z}_q \leftarrow$

$\rightarrow b \in \mathbb{Z}_q$

$\rightarrow d \in \mathbb{Z}_q$

$$d = a \cdot b - c$$

In this talk

- Intro and Where We Fit
- A Brief Intro to DKLs
    1. Rewriting ECDSA
    2. Consistency Machinery
    3. Secure Multiplication
- Implementations and Performance
- What's in our Submission

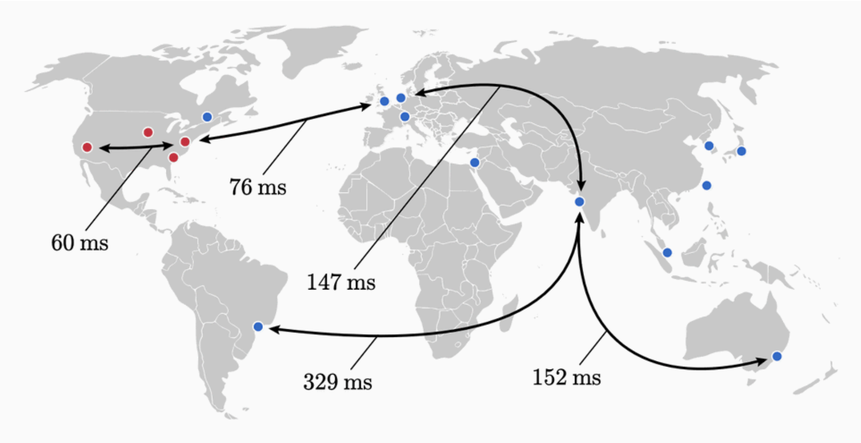
Authors' Proof of Concept

("academic code," 2023)

Language: Rust + Custom EC Library

Optimizations: Multithread + SHA-NI

Setting: Google Cloud 8-Thread



Results:

	2P	4P	16P	256P
LAN	4.0ms	8.4ms	40ms	686ms
US WAN		105ms	114ms	750ms
Global WAN			495ms	1,326ms

Silence Laboratories

(audited professional implementation, 2025)

Language: Rust + Standard Crypto Crates

Optimizations: Single-threaded, software-only

Results:

- Browser + phone
  (comm over internet)





70ms



90ms

- Cloud + phone





57ms



85ms

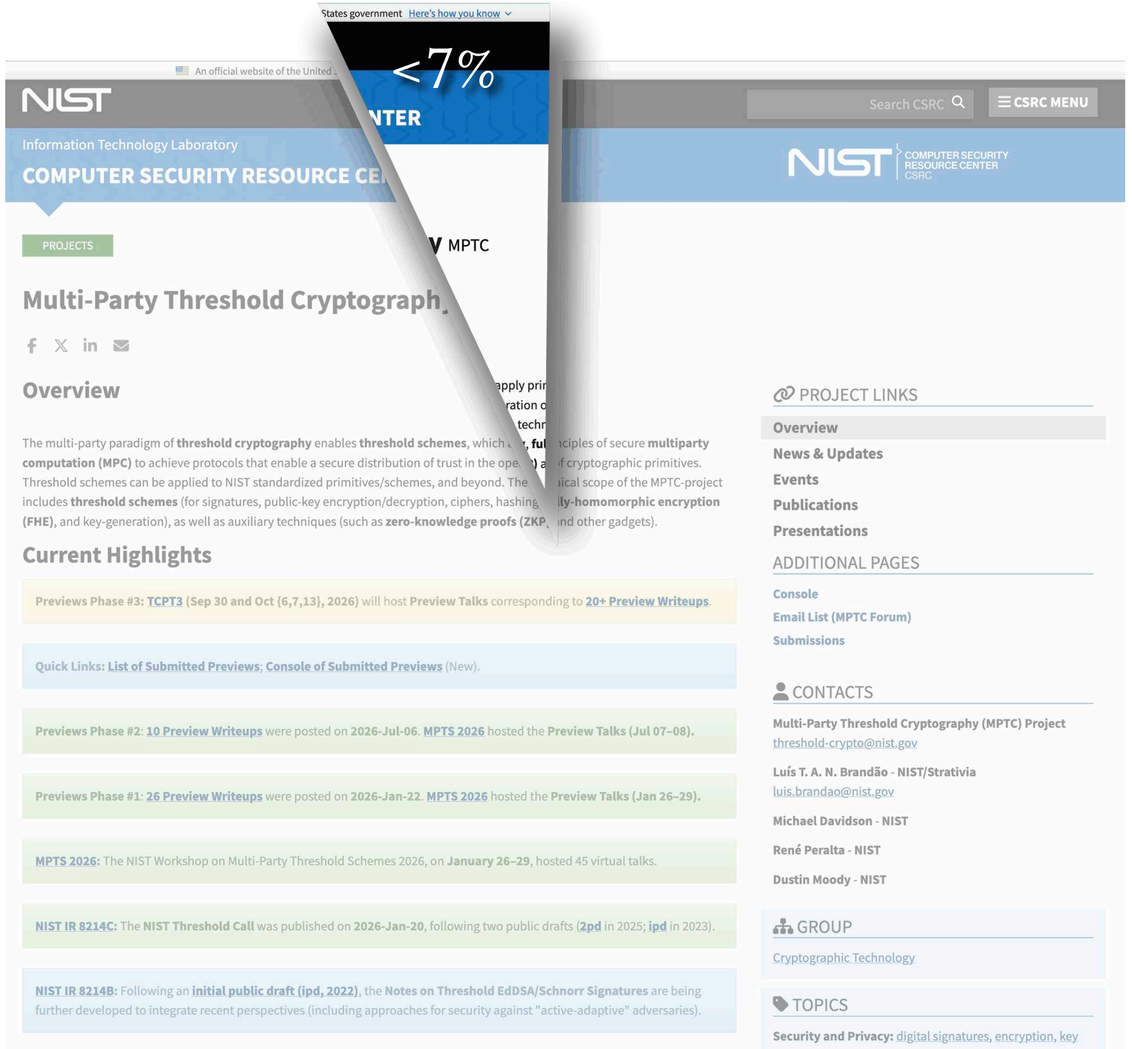
Source + Recent Benchmarks:
https://dkls23.silencelaboratories.com

In all cases, we signed over a 256-bit curve with statistical parameter 80.

Bandwidth

For a 256-bit curve,
statistical parameter 80:

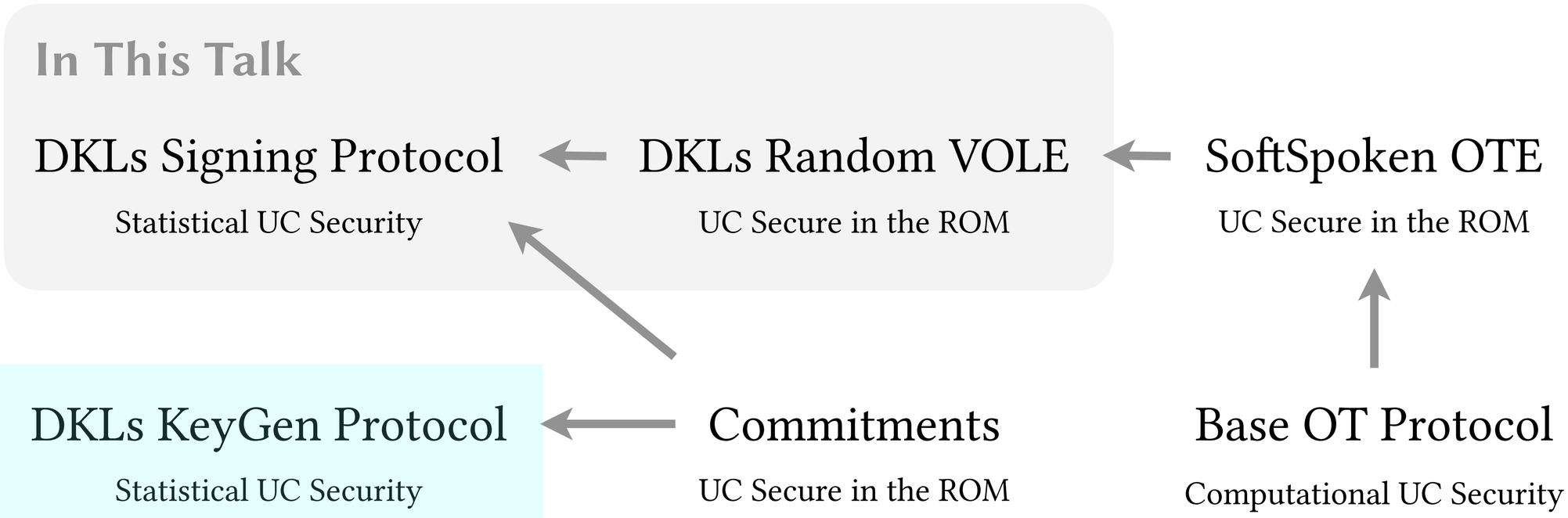
Each party sends
<55KB to every other party



In this talk

- Intro and Where We Fit
- A Brief Intro to DKLs
    1. Rewriting ECDSA
    2. Consistency Machinery
    3. Secure Multiplication
- Implementations and Performance
- What's in our Submission

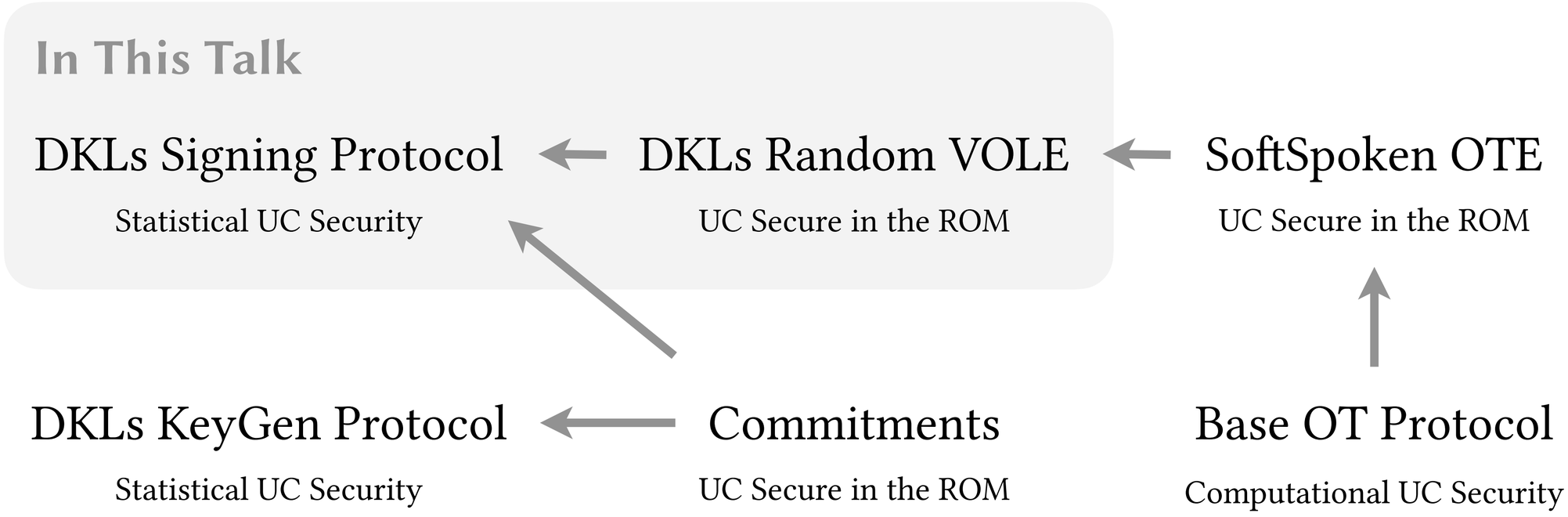
What's in Our Submission



Our signing is compatible with many KeyGen protocols

We propose one: 3 point-to-point rounds, commitments only, statistical security

What's in Our Submission



Additionally:

Parameterizations for every NIST-standardized curve and several levels of statistical security.

Machine-verifiable proofs of (*standalone*) security!

Thanks!

DKLs: Threshold ECDSA

More Information: http://dkls.info

Jack Doerner, NIST TCPT3, October 7 2026