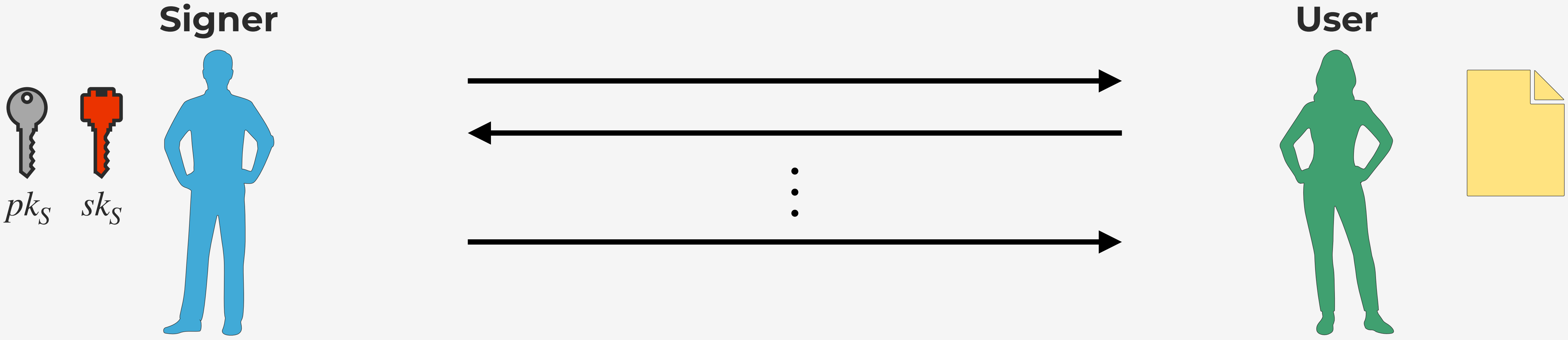
CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

Non-Interactive Blind Signatures from RSA Assumption and More

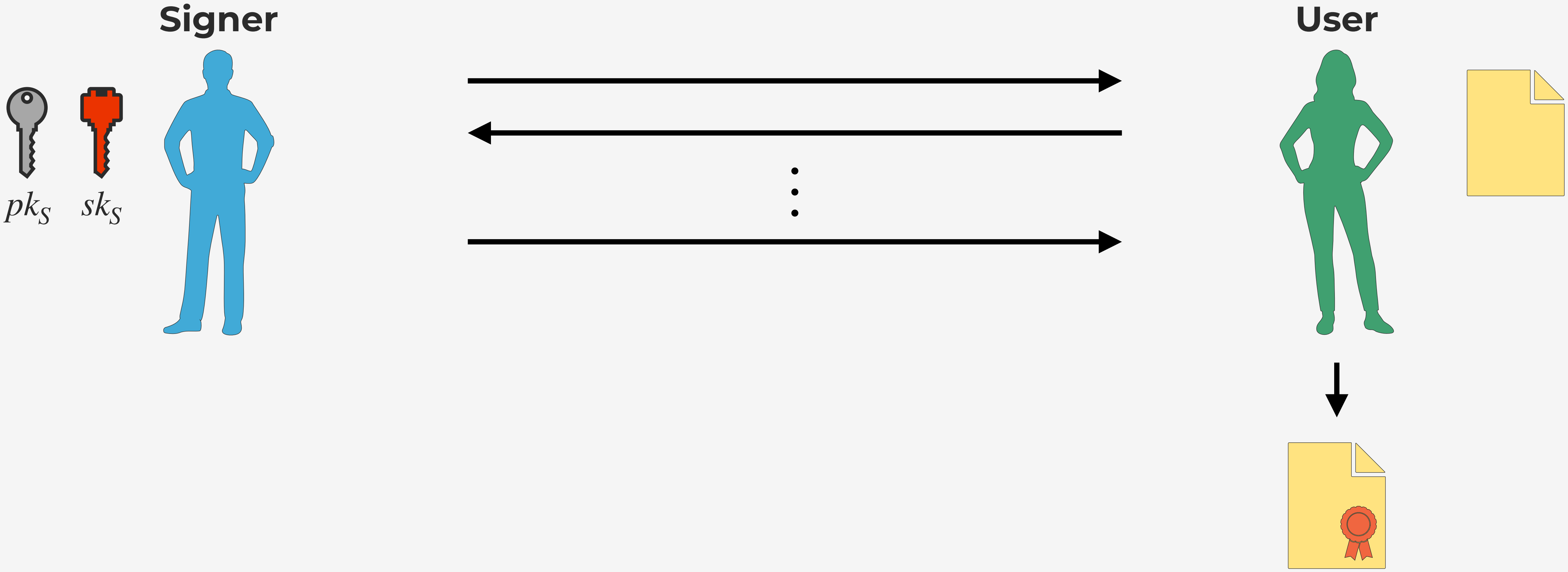
L. Hanzlik, E. Paracucchi, R. Zanotto

Eurocypt 2025 | Madrid | May 6th

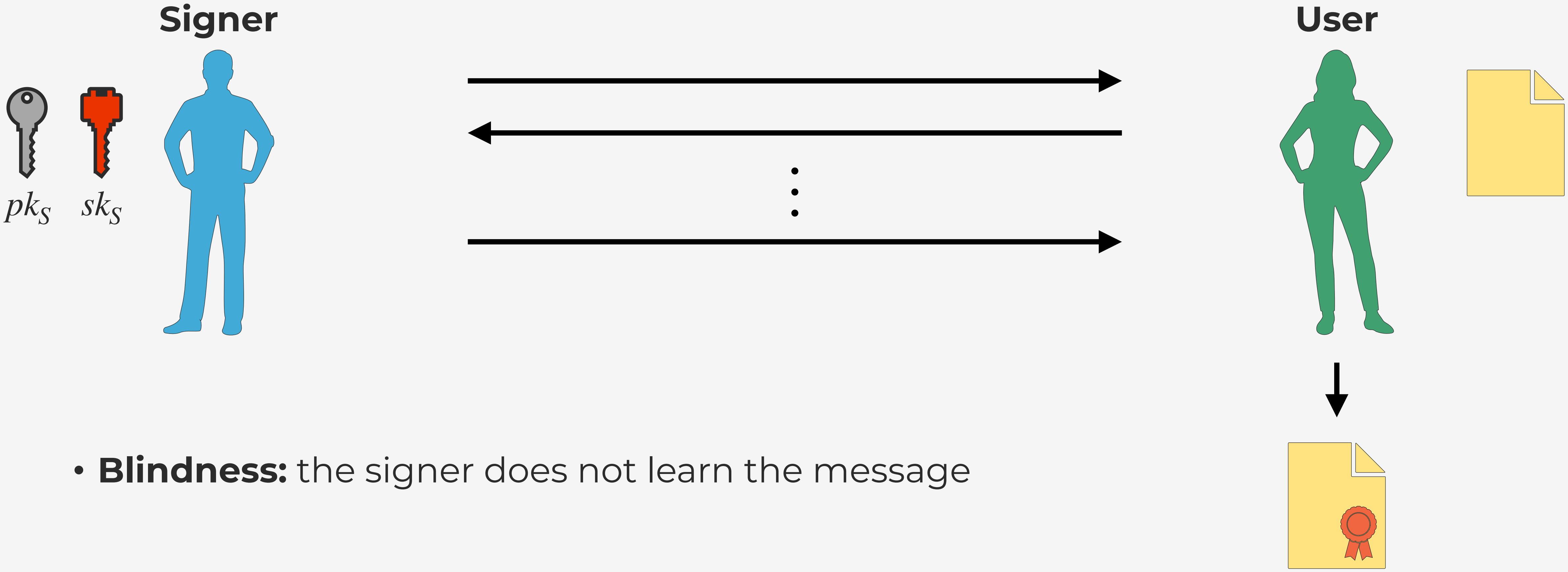
Blind Signatures



Blind Signatures

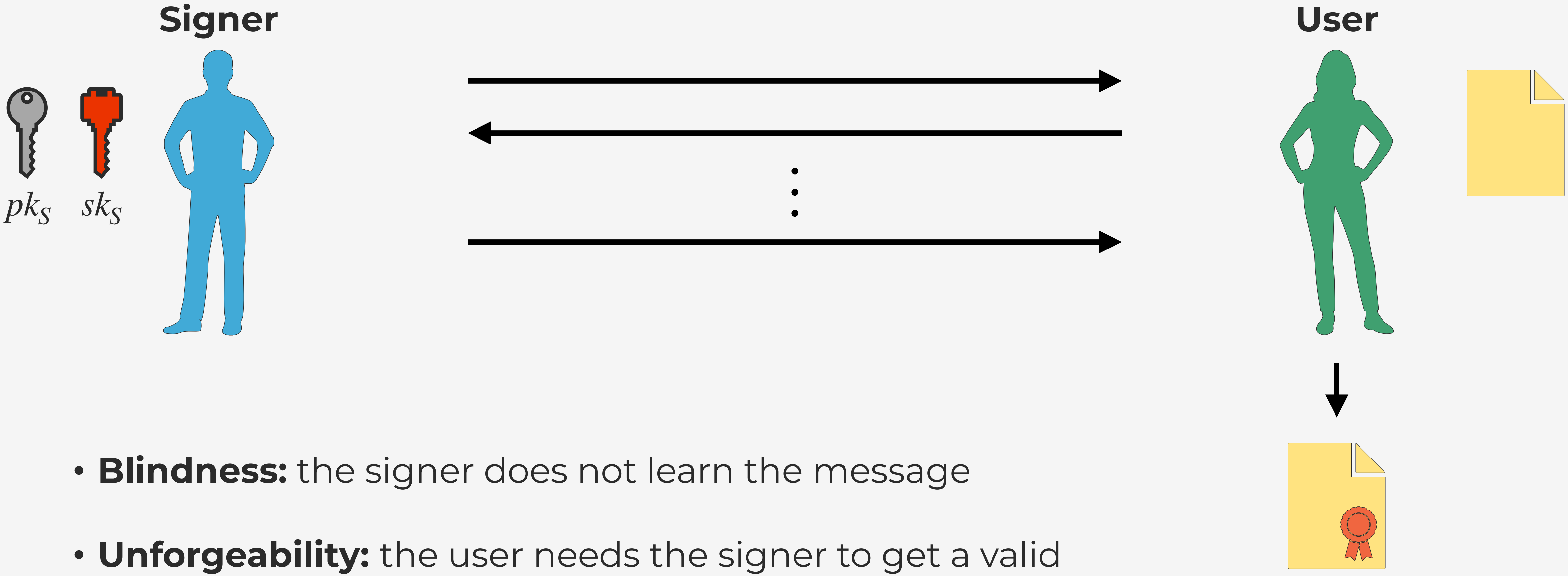


Blind Signatures



- **Blindness:** the signer does not learn the message

Blind Signatures



- **Blindness:** the signer does not learn the message
- **Unforgeability:** the user needs the signer to get a valid signature

Applications of BS

Introduced by David Chaum in 1980s. Used as **one-time anonymous tokens**

Applications of BS

Introduced by David Chaum in 1980s. Used as **one-time anonymous tokens**

- 💰 **e-Cash**

Applications of BS

Introduced by David Chaum in 1980s. Used as **one-time anonymous tokens**

- 💰 **e-Cash**
- 🪪 **Anonymous Credentials**

Applications of BS

Introduced by David Chaum in 1980s. Used as **one-time anonymous tokens**

- 💰 **e-Cash**
- 🪪 **Anonymous Credentials**
- 🎟️ **Privacy Pass**

Applications of BS

Introduced by David Chaum in 1980s. Used as **one-time anonymous tokens**

- 💰 **e-Cash**
- 🪪 **Anonymous Credentials**
- 🎟️ **Privacy Pass**

In many applications the message is just a **random string**

Applications of BS

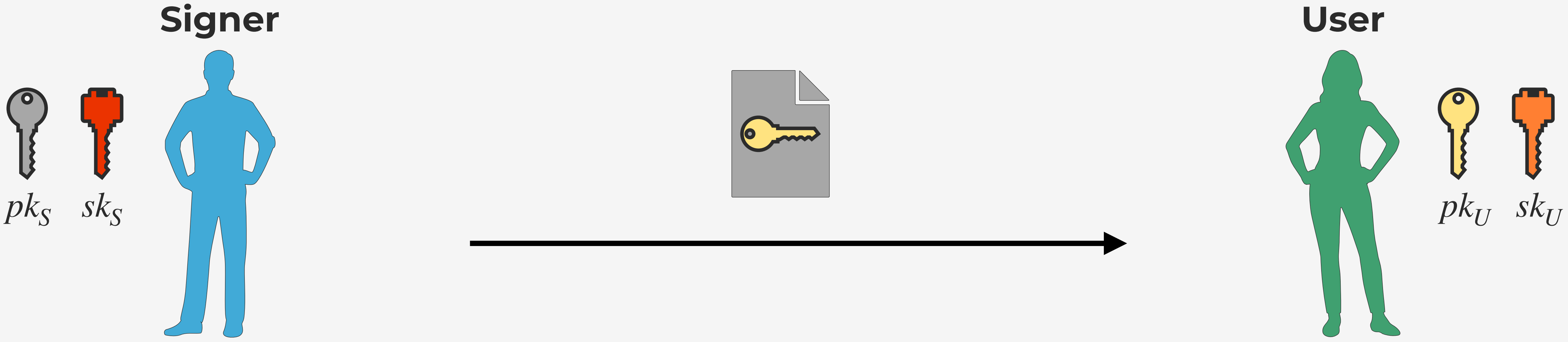
Introduced by David Chaum in 1980s. Used as **one-time anonymous tokens**

- 💰 **e-Cash**
- 🪪 **Anonymous Credentials**
- 🎟️ **Privacy Pass**

In many applications the message is just a **random string**

No need of interaction

Non-Interactive Blind Signatures [Han23]



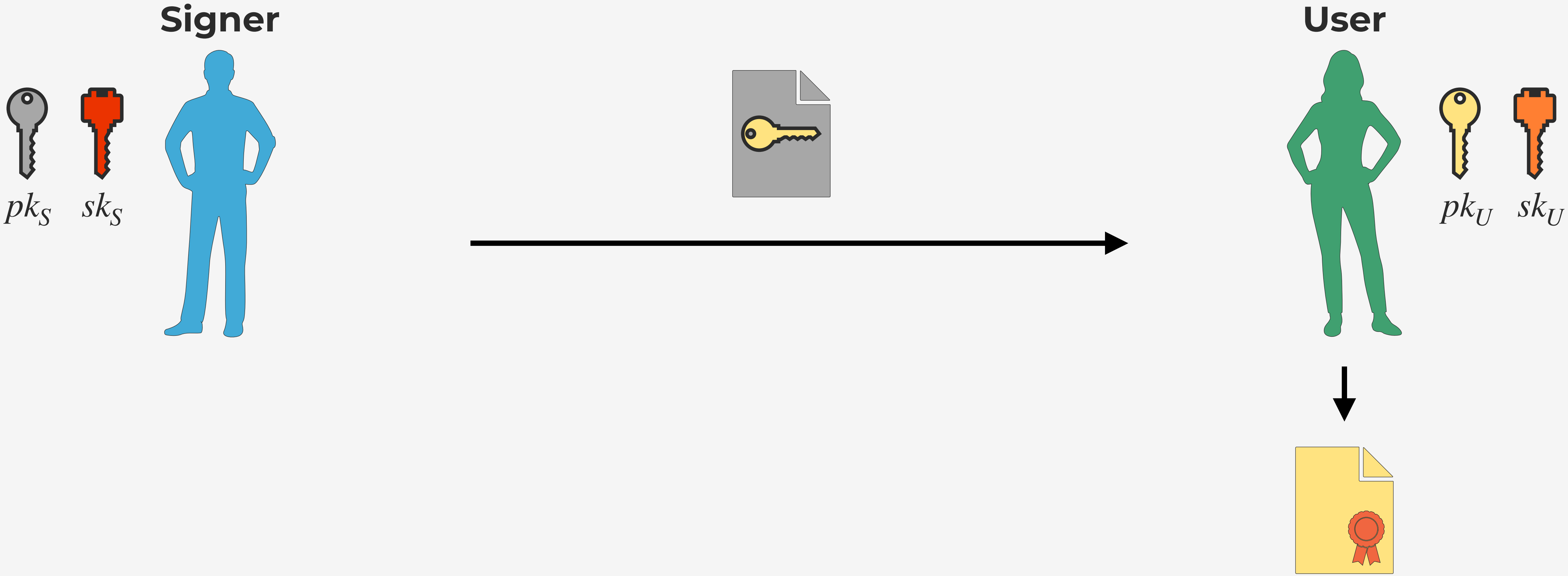
Signer

User

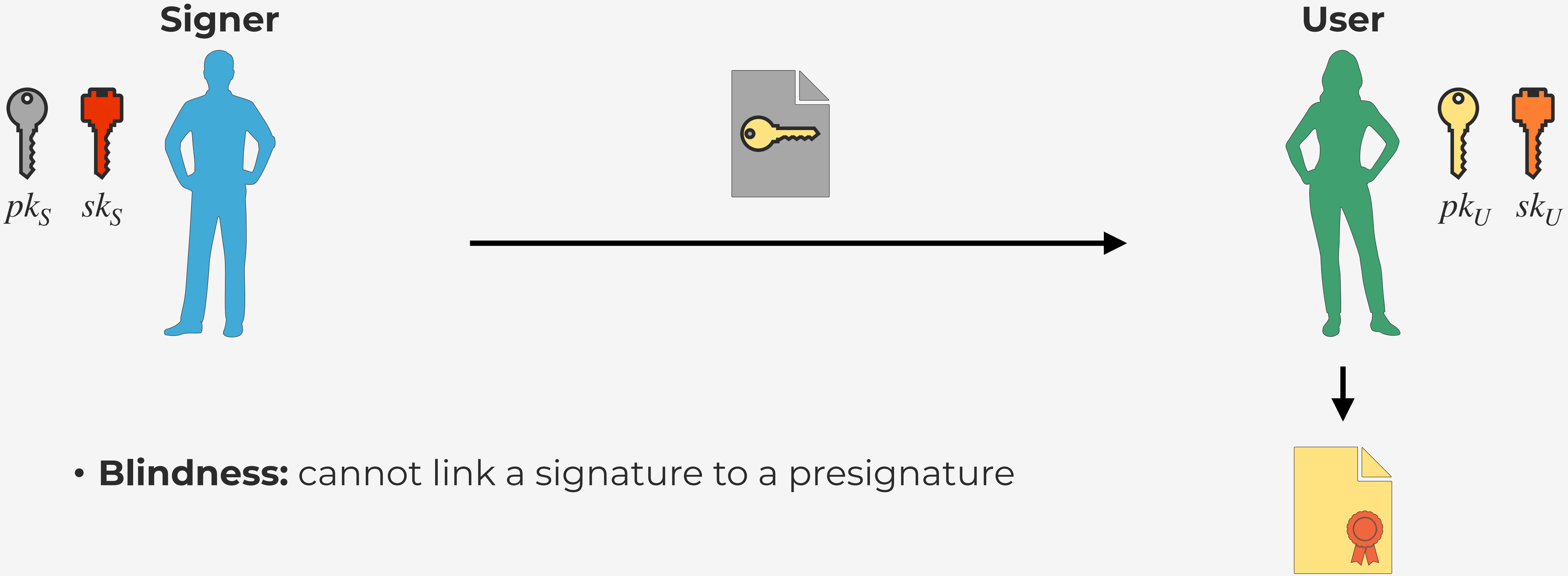
pk_S sk_S

pk_U sk_U

Non-Interactive Blind Signatures [Han23]

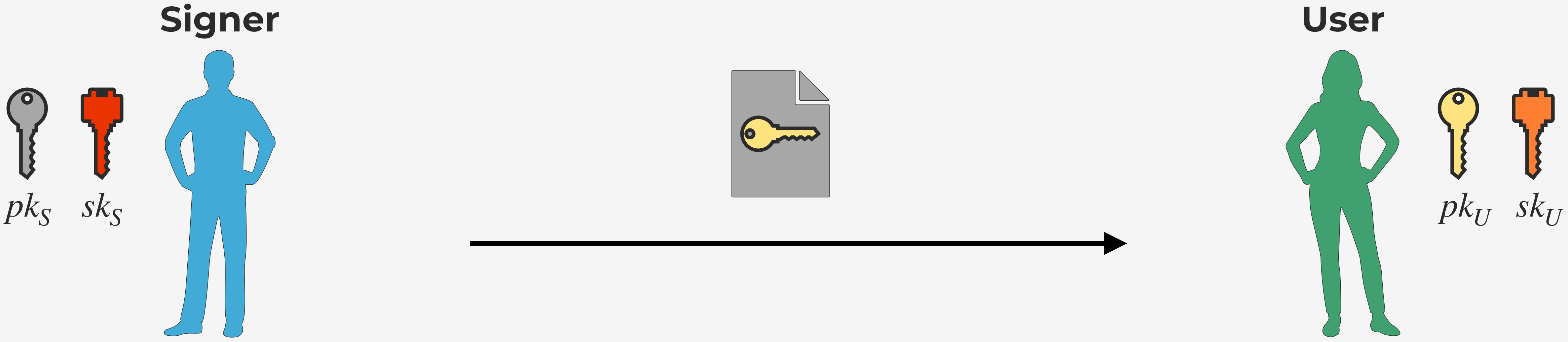


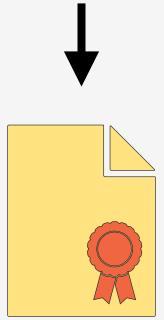
Non-Interactive Blind Signatures [Han23]



- **Blindness:** cannot link a signature to a presignature

Non-Interactive Blind Signatures [Han23]

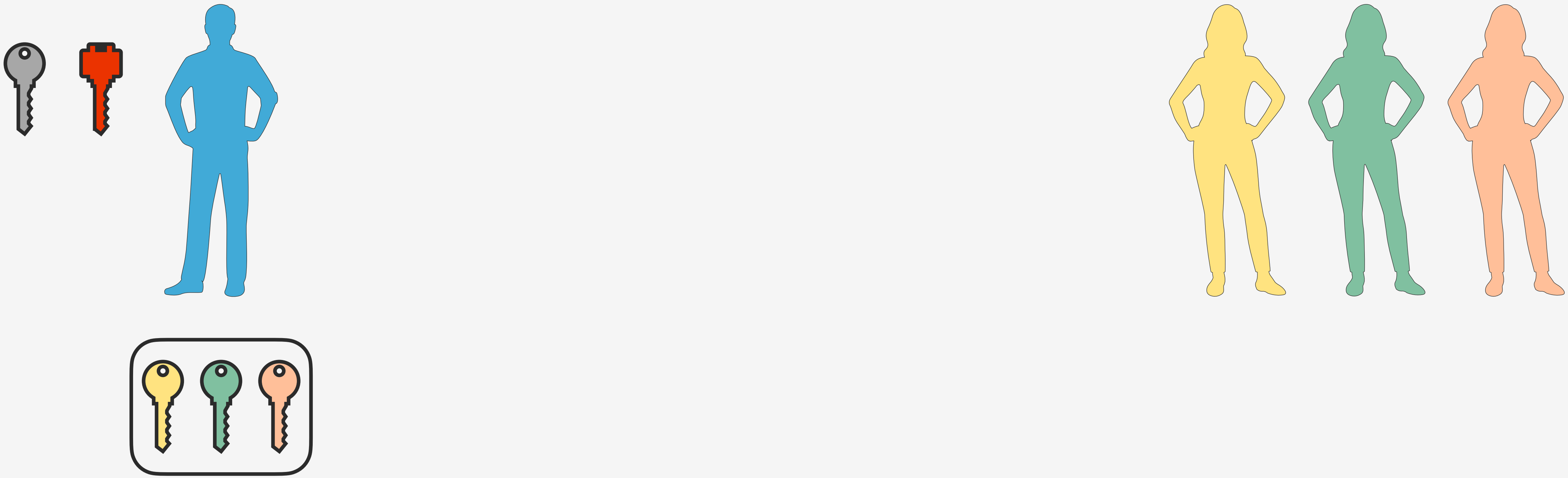




- **Blindness:** cannot link a signature to a presignature
- **Unforgeability:** cannot create $\ell + 1$ signatures from ℓ presignatures

Airdropping

Airdropping

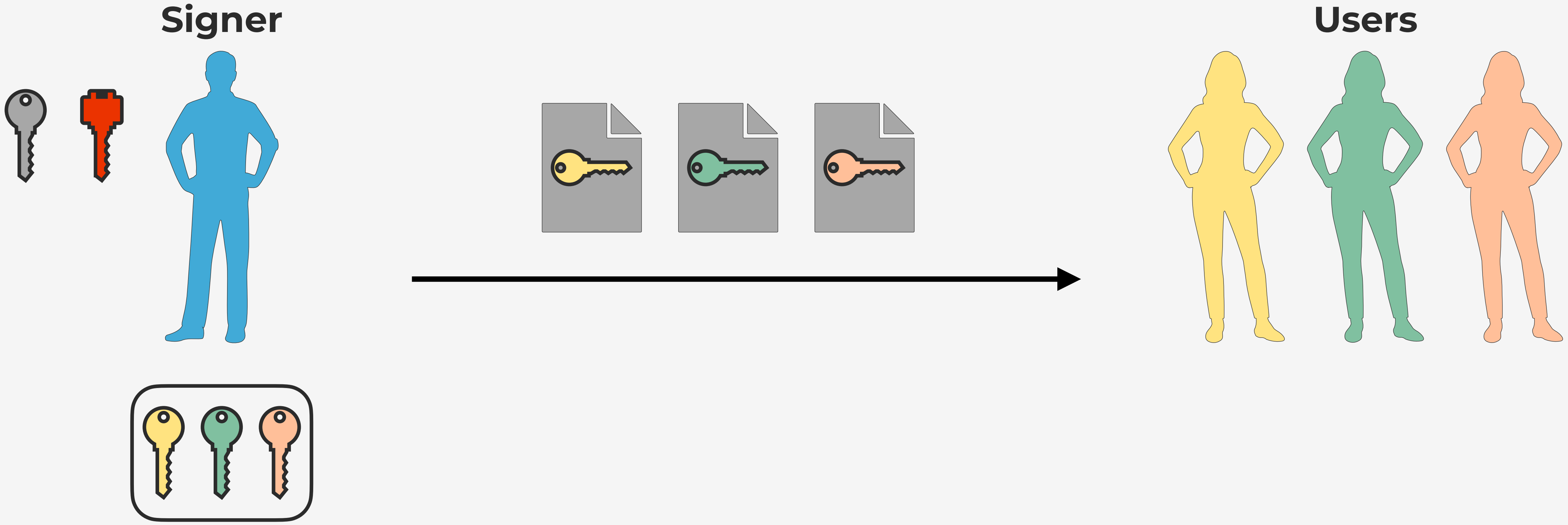


Signer

Users

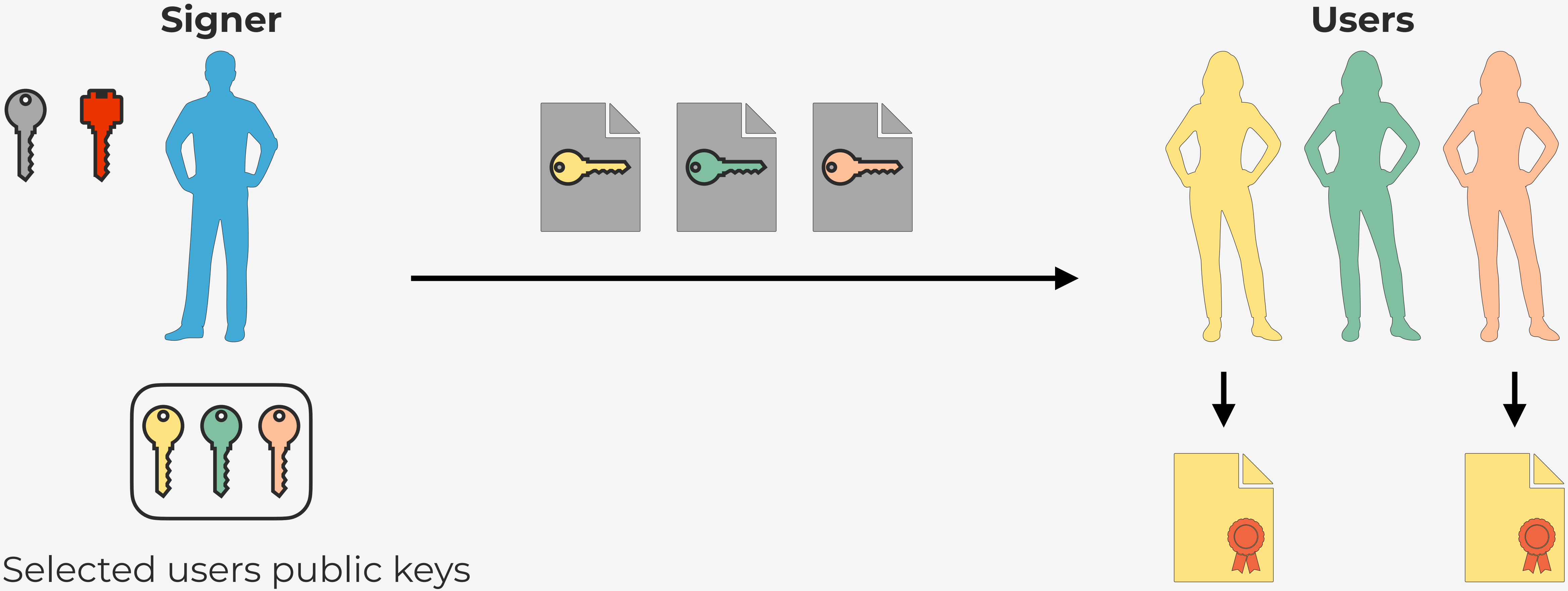
Selected users public keys

Airdropping



Selected users public keys

Airdropping



Selected users public keys

Challenges

The user’s public key corresponds to a long-term public key for other schemes such as GitHub public keys, PGP keys etc.

Challenges

The user’s public key corresponds to a long-term public key for other schemes such as GitHub public keys, PGP keys, etc.

The scheme proposed in [Han23] uses **specific keys**; users need to generate ad hoc keys

Challenges

The user’s public key corresponds to a long-term public key for other schemes such as GitHub public keys, PGP keys etc.

The scheme proposed in [Han23] uses **specific keys**; users need to generate ad hoc keys

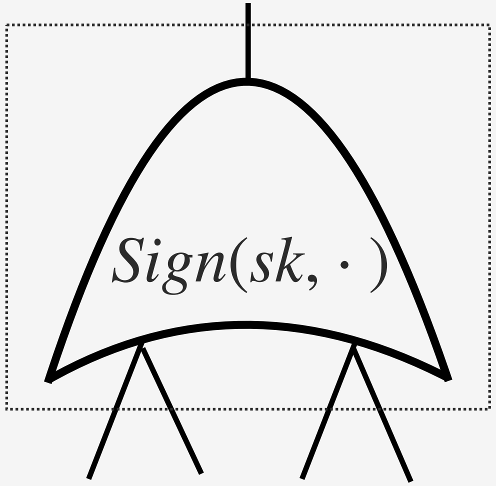
This work: construction of a NIBS compatible with standard RSA keys (N, e)

2PC and Yao’s Garbled Circuits

We need some way for the signer to send obliviously some data to the user

2PC and Yao's Garbled Circuits

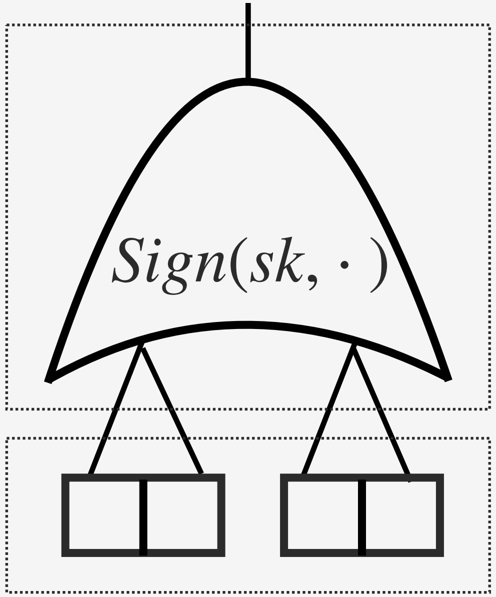
We need some way for the signer to send obliviously some data to the user



→ **Garbled encoding** of the fuction $Sign(sk, \cdot\,)$

2PC and Yao's Garbled Circuits

We need some way for the signer to send obliviously some data to the user

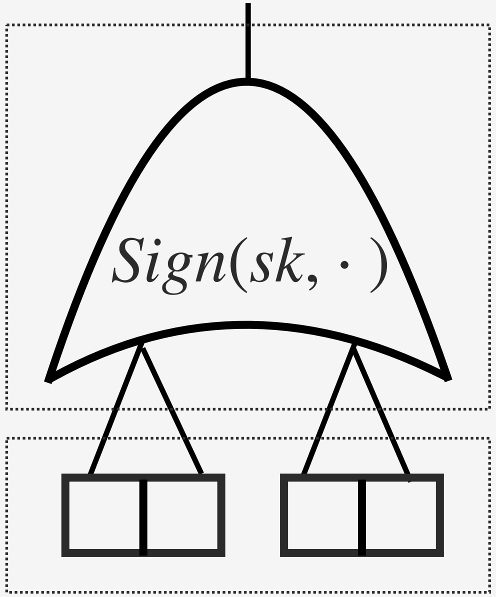


→ **Garbled encoding** of the fuction $Sign(sk, \cdot\ )$

→ Labels for the circuit sent using **oblivious transfer**

2PC and Yao’s Garbled Circuits

We need some way for the signer to send obliviously some data to the user



→ **Garbled encoding** of the fuction $Sign(sk, \cdot\,)$

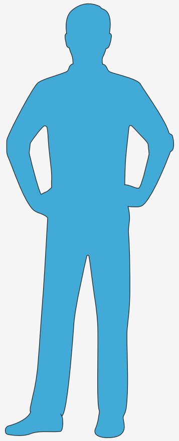
→ Labels for the circuit sent using **oblivious transfer**

The construction is interactive!

Non-Interactive Oblivious Transfer (NIOT)

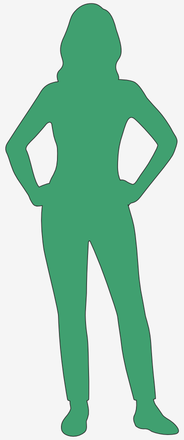
Sender

m_0, m_1, pk

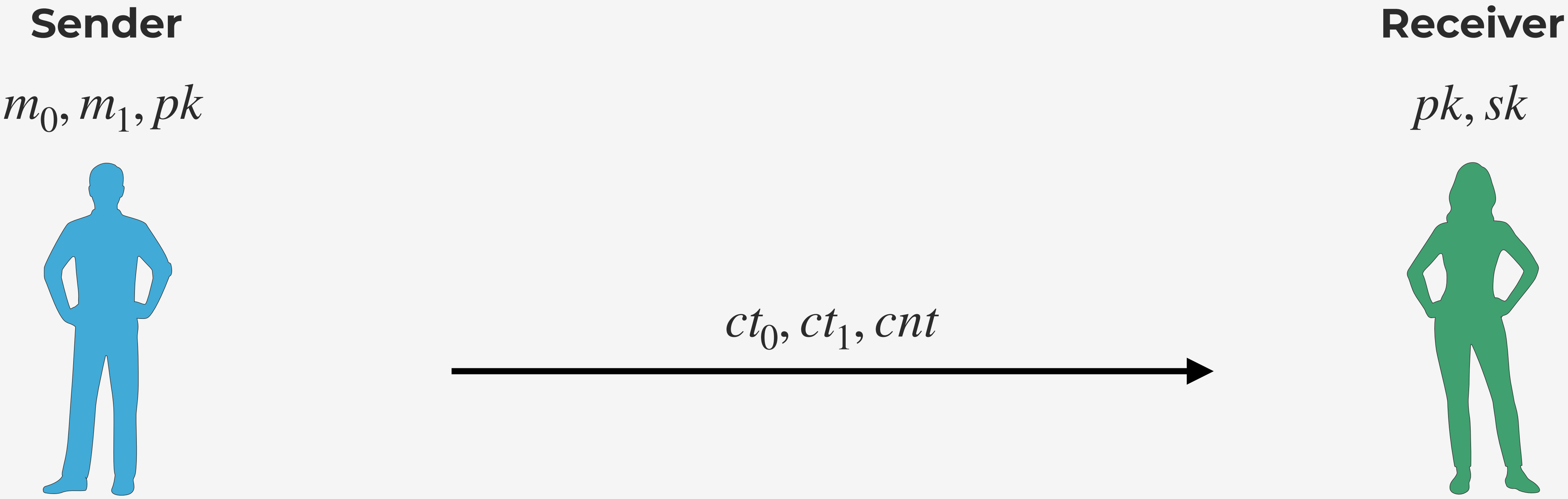


Receiver

pk, sk



Non-Interactive Oblivious Transfer (NIOT)



Sender

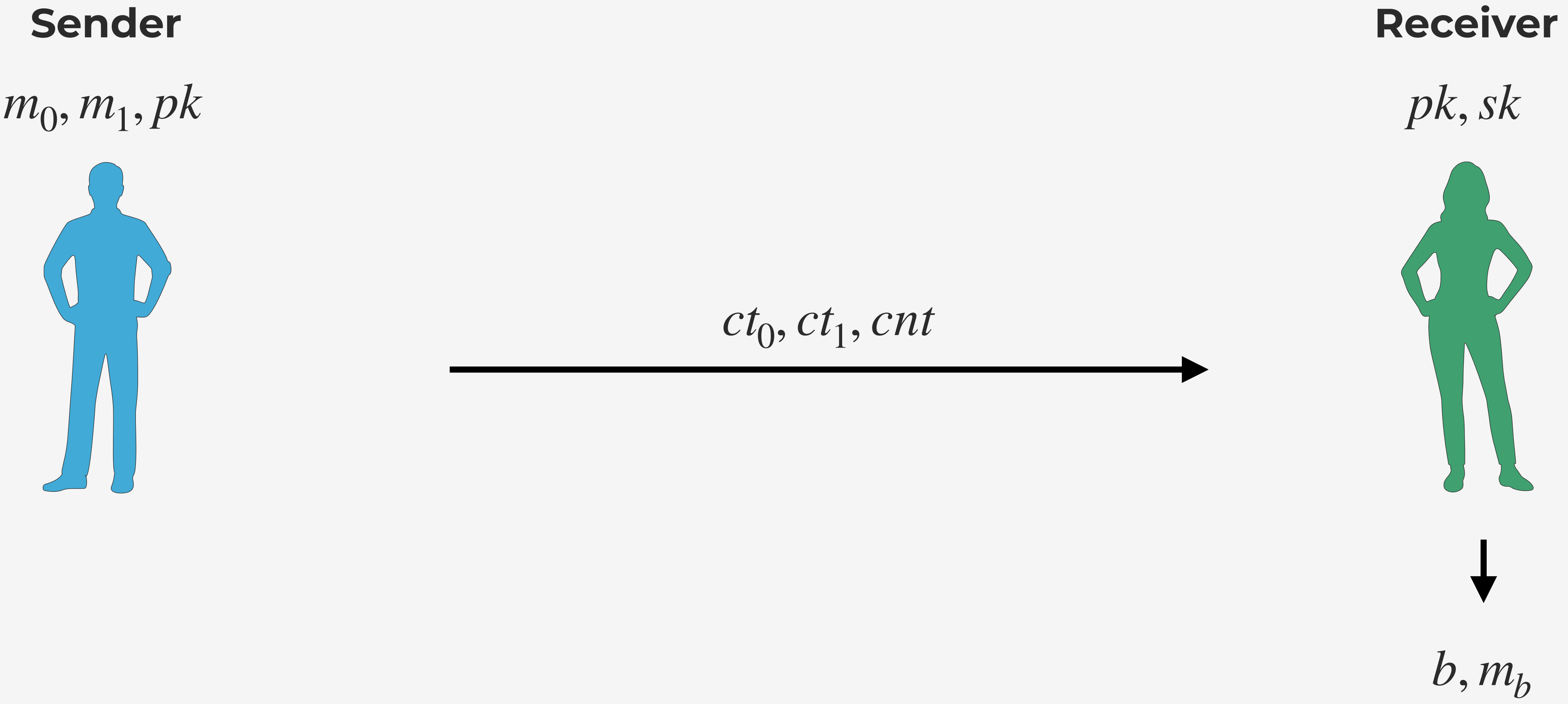
m_0, m_1, pk

Receiver

pk, sk

ct_0, ct_1, cnt

Non-Interactive Oblivious Transfer (NIOT)



Sender

m_0, m_1, pk

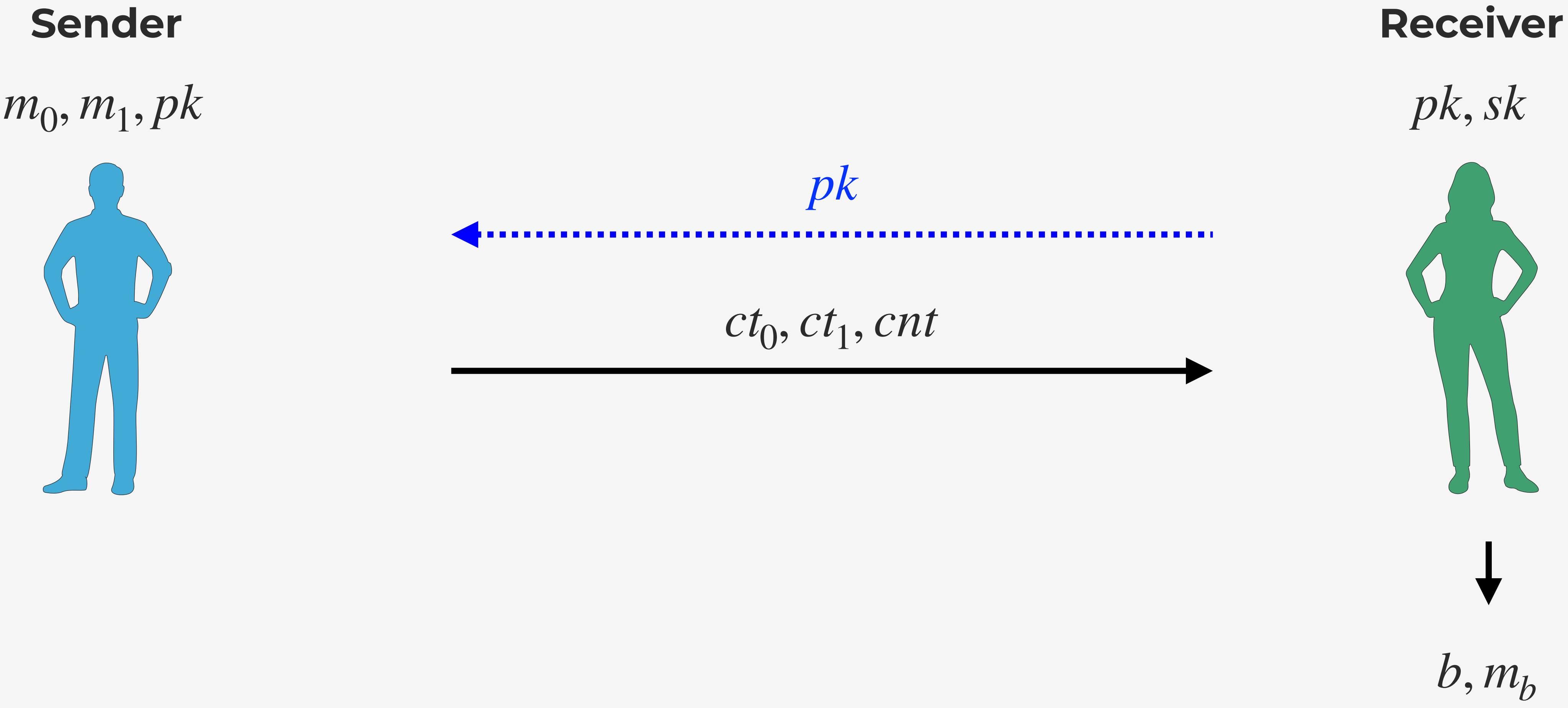
ct_0, ct_1, cnt

Receiver

pk, sk

b, m_b

Non-Interactive Oblivious Transfer (NIOT)



Sender

m_0, m_1, pk

Receiver

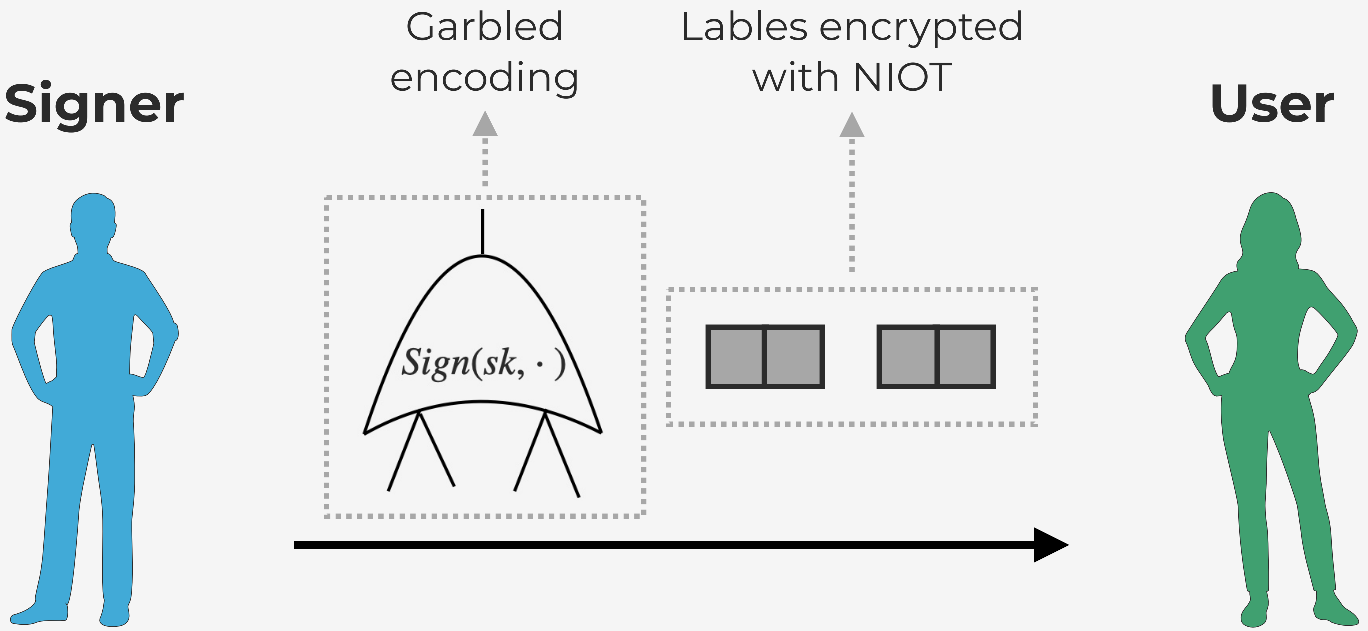
pk, sk

pk

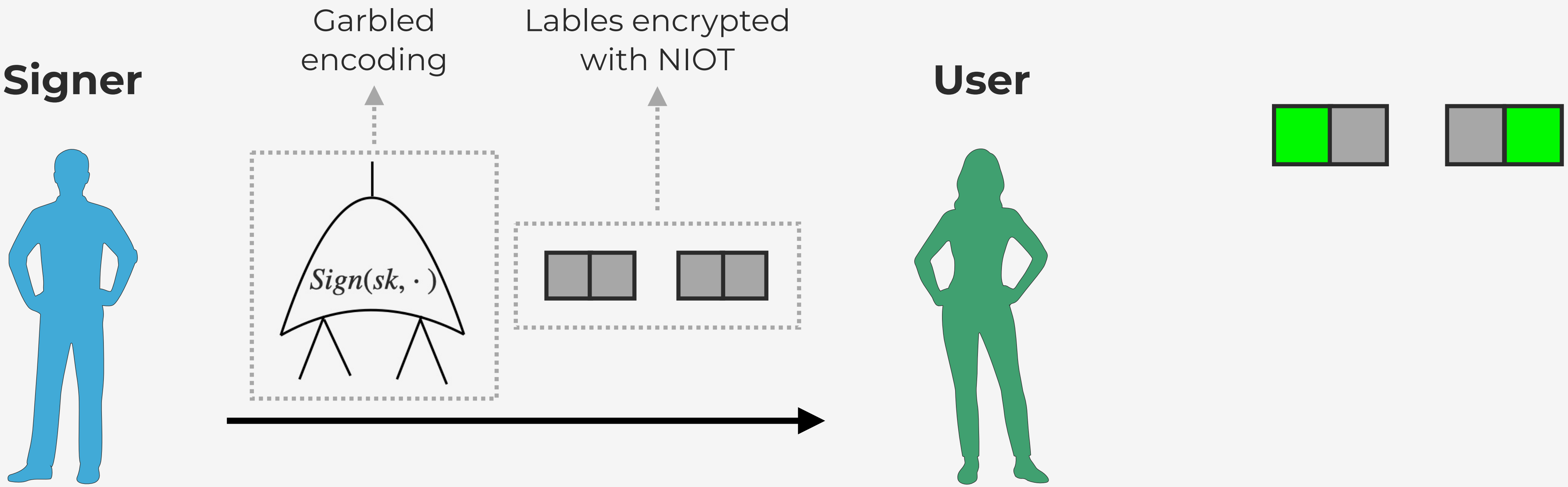
ct_0, ct_1, cnt

b, m_b

Non-Interactive Blind Signature



Non-Interactive Blind Signature



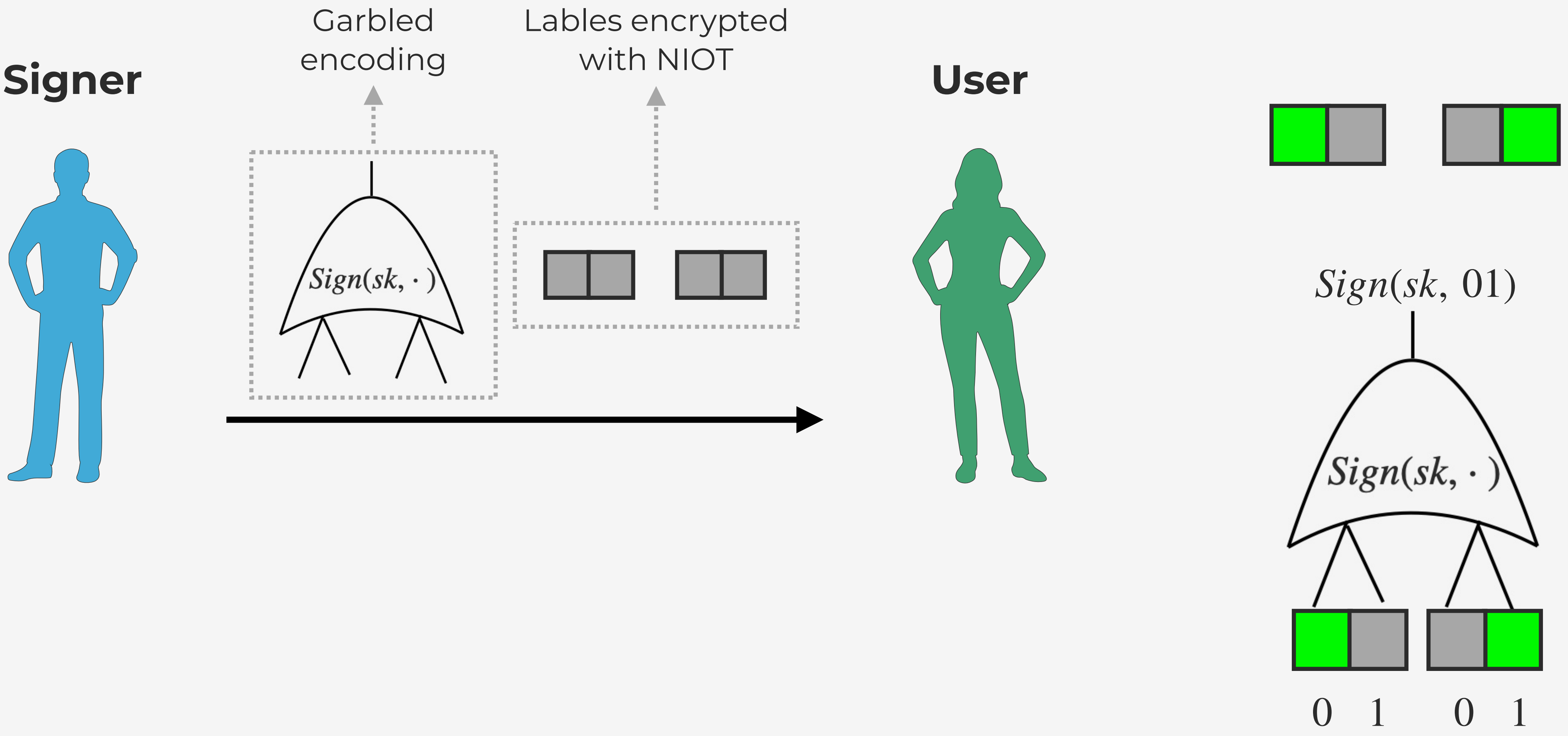
Signer

Garbled encoding

Lables encrypted with NIOT

User

Non-Interactive Blind Signature



Signer

Garbled encoding

Lables encrypted with NIOT

User

Our Contribution

🧰 Generic semi-honest NIBS from Yao’s GC + NIOT

Our Contribution

🧰 Generic semi-honest NIBS from Yao’s GC + NIOT

⚙️ Efficient garbling of signing functions

- Pointcheval-Sanders signatures
- RSA based signaturers

Our Contribution

🧰 Generic semi-honest NIBS from Yao’s GC + NIOT

⚙️ Efficient garbling of signing functions

- Pointcheval-Sanders signatures
- RSA based signaturers

🔗 Construction of NIOT supporting RSA keys

Our Contribution

🧰 Generic semi-honest NIBS from Yao’s GC + NIOT

⚙️ Efficient garbling of signing functions

- Pointcheval-Sanders signatures
- RSA based signaturers

🔗 Construction of NIOT supporting RSA keys

😈 Fully malicious NIBS for PS signature (non generic)

Our Contribution

🧰 Generic semi-honest NIBS from Yao’s GC + NIOT

⚙️ Efficient garbling of signing functions

- Pointcheval-Sanders signatures
- RSA based signaturers

🔗 Construction of NIOT supporting RSA keys → Rest of this talk

😈 Fully malicious NIBS for PS signature (non generic)

Quadratic Residuosity

The public key $N = pq$ must somehow encode the user’s choice in a way that the sender cannot distinguish

Quadratic Residuosity

The public key $N = pq$ must somehow encode the user's choice in a way that the sender cannot distinguish

Quadratic residuosity problem: decide whether an element $x \in \mathbb{Z}_N$ with Jacobi symbol 1 is a quadratic residue or a quadratic non-residue

Quadratic Residuosity

The public key $N = pq$ must somehow encode the user's choice in a way that the sender cannot distinguish

Quadratic residuosity problem: decide whether an element $x \in \mathbb{Z}_N$ with Jacobi symbol 1 is a quadratic residue or a quadratic non-residue

$m_0 \iff$ square $\qquad$ $m_1 \iff$ non-square

Goldwasser-Micali vs Cocks

Goldwasser-Micali vs Cocks

pk : $N, x \in \mathbb{Z}_N$ non-square

sk : factorization of N

Goldwasser-Micali vs Cocks

$pk : N, x \in \mathbb{Z}_N$ non-square

sk : factorization of N

Hiding property

If x is a square, $GM.Enc(x, m)$ statistically hides m

Goldwasser-Micali vs Cocks

$pk : N, x \in \mathbb{Z}_N$ non-square

sk : factorization of N

Hiding property

If x is a square, $GM.Enc(x, m)$ statistically hides m

$pk : N, x \in \mathbb{Z}_N$ square

sk : factorization of N, s s.t. $s^2 \equiv x \pmod{N}$

Goldwasser-Micali vs Cocks

pk : $N, x \in \mathbb{Z}_N$ non-square

sk : factorization of N

Hiding property

If x is a square, $GM.Enc(x, m)$ statistically hides m

pk : $N, x \in \mathbb{Z}_N$ square

sk : factorization of N, s s.t. $s^2 \equiv x \pmod{N}$

Hiding property

If x is a non-square, and N is squarefree

then $Cocks.Enc(x, m)$ statistically hides m

NIOT Construction, squarefree modulus

Sender

$m_0, m_1, pk = N$

Receiver

$pk = N, sk = fact(N)$

NIOT Construction, squarefree modulus

Sender

$m_0, m_1, pk = N$

Receiver

$pk = N, sk = fact(N)$

$x \leftarrow \mathscr{H}_N(N, cnt)$

$ct_0 \leftarrow Cocks.Enc(x, m_0)$

$ct_1 \leftarrow GM.Enc(x, m_1)$

$$\xrightarrow{\quad ct_0, ct_1, cnt \quad}$$

NIOT Construction, squarefree modulus

Sender

$m_0, m_1, pk = N$

$x \leftarrow \mathscr{H}_N(N, cnt)$

$ct_0 \leftarrow Cocks.Enc(x, m_0)$

$ct_1 \leftarrow GM.Enc(x, m_1)$

Receiver

$pk = N, sk = fact(N)$

$$\xrightarrow{\quad ct_0, ct_1, cnt \quad}$$

$x \leftarrow \mathscr{H}(N, cnt)$

if x is a square:

$m \leftarrow Cocks.Dec(x, ct_0)$

else:

$m \leftarrow GM.Dec(x, ct_1)$

Generic Modulus

If N is not squarefree then a malicious receiver might decrypt both ciphertexts

Generic Modulus

If N is not squarefree then a malicious receiver might decrypt both ciphertexts

Idea: encrypt the ciphertext ct_0, ct_1 with a key k that can be recovered only if N is squarefree

Generic Modulus

If N is not sqf $\Rightarrow d = gcd(N, \phi(N)) > 1 \Rightarrow$ the equation $X^N = a$ has (zero or) at least $d > 1$ solutions in $\mathbb{Z}_N^*$

Generic Modulus

If N is not sqf $\Rightarrow d = gcd(N, \phi(N)) > 1 \Rightarrow$ the equation $X^N = a$ has (zero or) at least $d > 1$ solutions in $\mathbb{Z}_N^*$

Sender: sample $a_1, \ldots, a_\lambda \leftarrow \mathbb{Z}_N$ and derive a key $k = \mathscr{H}(a_1, \ldots, a_\lambda)$, encrypt the ciphertexts with k and send them along with $b_i := a_i^N$ for $i = 0, \ldots, \lambda$

Generic Modulus

If N is not sqf $\Rightarrow d = gcd(N, \phi(N)) > 1 \Rightarrow$ the equation $X^N = a$ has (zero or) at least $d > 1$ solutions in $\mathbb{Z}_N^*$

Sender: sample $a_1, \ldots, a_\lambda \leftarrow \mathbb{Z}_N$ and derive a key $k = \mathscr{H}(a_1, \ldots, a_\lambda)$, encrypt the ciphertexts with k and send them along with $b_i := a_i^N$ for $i = 0, \ldots, \lambda$

- If N is sqf then w.h.p. $gcd(N, \phi(N)) = 1$ then the receiver can recover from b_i's the (unique) a_i's and hence k

Generic Modulus

If N is not sqf $\Rightarrow d = gcd(N, \phi(N)) > 1 \Rightarrow$ the equation $X^N = a$ has (zero or) at least $d > 1$ solutions in $\mathbb{Z}_N^*$

Sender: sample $a_1, \ldots, a_\lambda \leftarrow \mathbb{Z}_N$ and derive a key $k = \mathscr{H}(a_1, \ldots, a_\lambda)$, encrypt the ciphertexts with k and send them along with $b_i := a_i^N$ for $i = 0, \ldots, \lambda$

- If N is sqf then w.h.p. $gcd(N, \phi(N)) = 1$ then the receiver can recover from b_i's the (unique) a_i's and hence k
- Otherwise the equation $X^N = b_i$ has more than d solutions, hence will recover the right a_i with probability less than $1/d$. Therefore a malicious can decrypt with probability less than $(1/d)^\lambda$

NIOT Construction, generic modulus

Sender

$m_0, m_1, pk = N$

$x \leftarrow \mathscr{H}_N(N, cnt)$

$ct_0 \leftarrow Cocks.Enc(x, m_0)$

$ct_1 \leftarrow GM.Enc(x, m_1)$

$a_1, \ldots, a_\lambda \leftarrow \mathbb{Z}_N$

$k \leftarrow \mathscr{H}(a_1, \ldots, a_\lambda)$

$b_i \leftarrow a_i^N$ for $i = 0, \ldots, \lambda$

$$\xrightarrow{\begin{array}{c} Enc_k(ct_0), Enc_k(ct_1), cnt, \\ b_1, \ldots, b_\lambda \end{array}}$$

Receiver

$pk = N, sk = fact(N)$

From b_i recover k

Recover ct_0, ct_1

$x \leftarrow \mathscr{H}(N, cnt)$

if x is a square:

$m \leftarrow Cocks.Dec(x, ct_0)$

else:

$m \leftarrow GM.Dec(x, ct_1)$

Wrapping it up

⭐ We have built a **NIOT supporting RSA user’s public key**

This gives us, combined with efficient garbling of signing functions, the **first NIBS compatible with standard RSA keys**

Wrapping it up

⭐ We have built a **NIOT supporting RSA user's public key**

This gives us, combined with efficient garbling of signing functions, the **first NIBS compatible with standard RSA keys**

🧩 **Generic paradigm** to constuct NIBS

Wrapping it up

⭐ We have built a **NIOT supporting RSA user's public key**

This gives us, combined with efficient garbling of signing functions, the **first NIBS compatible with standard RSA keys**

🧩 **Generic paradigm** to constuct NIBS

🔮 Future work:

- Post-quantum construction
- More applications

PS Signature

Public parameters: p, G_1, G_2, G_T, e

$KeyGen$: sample $g \leftarrow G_2$ and $(x, y) \leftarrow \mathbb{Z}_p^2$, set $X = g^x$ and $Y = g^y$. Return the pair $pk = (X, Y),\ sk = (x, y)$

$Sign(sk, m)$: sample $h \leftarrow G_1$ and output $\sigma = (h, h^{x+ym})$

$Verify(pk, m, \sigma)$: parse $\sigma = (\sigma_1, \sigma_2)$ and check if $e(\sigma_1, X \cdot Y^m) = e(\sigma_2, g)$

Garbling PS Signature

We garble the second component σ_2

Let $\ell = \lfloor \log p \rfloor$, we consider the binary decomposition of $m = m_1 \ldots m_\ell$

Compute $a_1, \ldots, a_\ell \in G_1$ such that $\prod_{i=0}^{\ell} a_i = 1_{G_1}$, set $d = a_0 \cdot h^x$

For $i = 1, \ldots, \ell$ define $s_i^0 = a_i$ and $s_i^1 = a_i \cdot h^{2^{i-1}y}$

Derive ciphertexts $ct_i^0 = Enc(k_i^0, s_i^0)$ and $ct_i^1 = Enc(k_i^1, s_i^1)$ for some keys k_i^0, k_i^1

Garbled function: $\{ct_i^0, ct_i^1\}_i, h, d$

Labels: $\{k_i^0, k_i^1\}_i$