CISPA
Center for IT-Security, Privacy
and Accountability

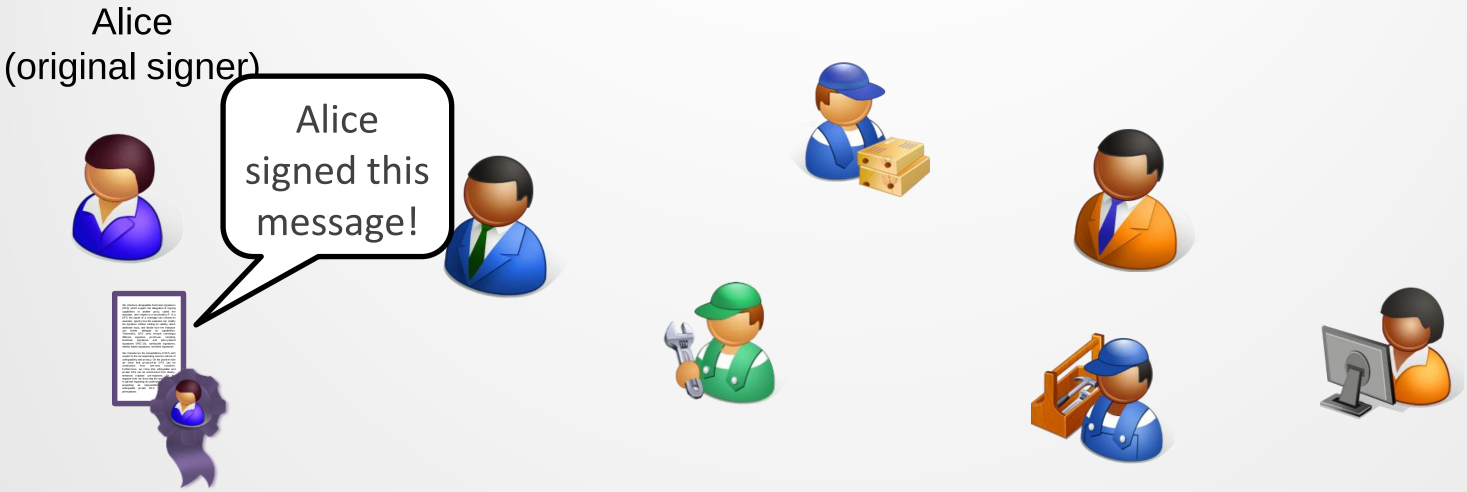
UNIVERSITÄT
DES
SAARLANDES

Delegatable Functional Signatures

Michael Backes, **Sebastian Meiser**, Dominique Schröder

Public Key Cryptography, March 7, 2016, Taipei

What is a malleable Signature?



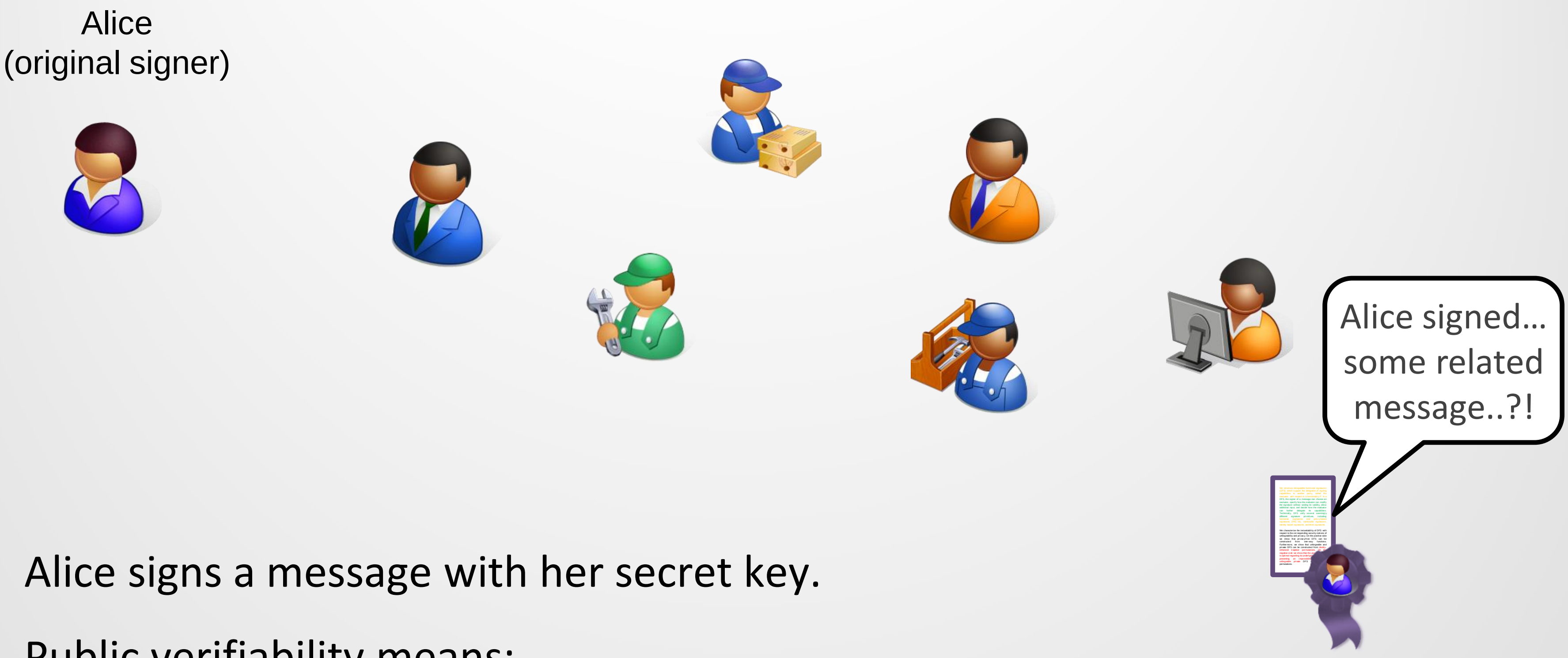
- Alice signs a message with her secret key.
- Public verifiability means:

a) Alice signed the message, or

b) Alice signed the message and the message has been modified, s.t. …

- … the resulting message still is in some relation to the signed message.
- … all operations performed on the message were “valid”.

What is a malleable Signature?



- Alice signs a message with her secret key.
- Public verifiability means:

a) Alice signed the message, or

b) Alice signed the message and the message has been modified, s.t. …

  - … the resulting message still is in some relation to the signed message.
  - … all operations performed on the message were "valid".

(Malleable) Signature Primitives

Homomorphic Signatures

Classical Signatures

Redactable Signatures

Rerandomizable Signatures

Proxy Signatures

Identity-based Signatures

Sanitizable Signatures

Blind Signatures

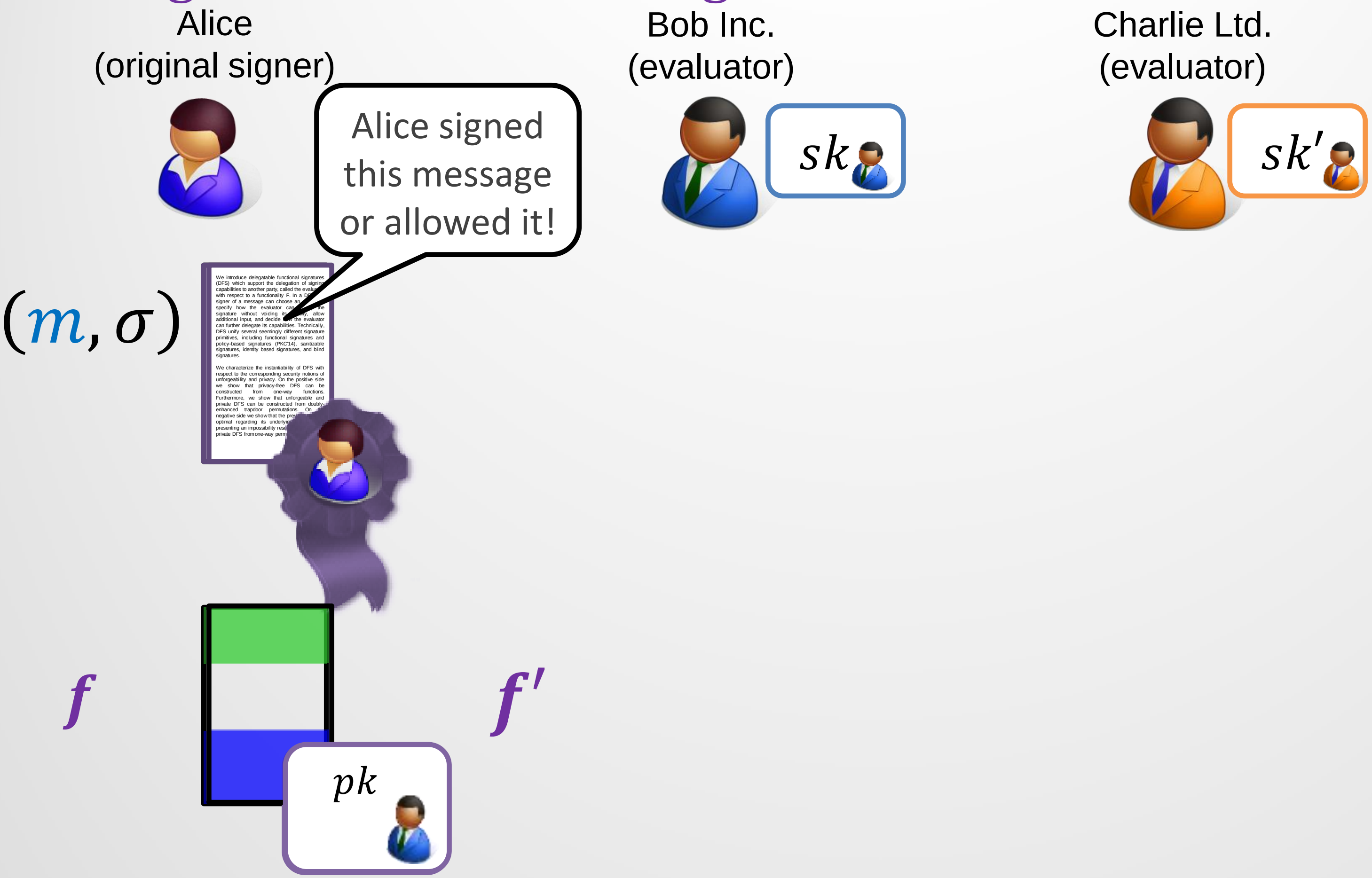
Functional Digital Signatures [BGI]

PKC'15

Policy-based Signatures [BF]

Goal: Generalization and simplification of primitives and notions

Delegatable Functional Signatures

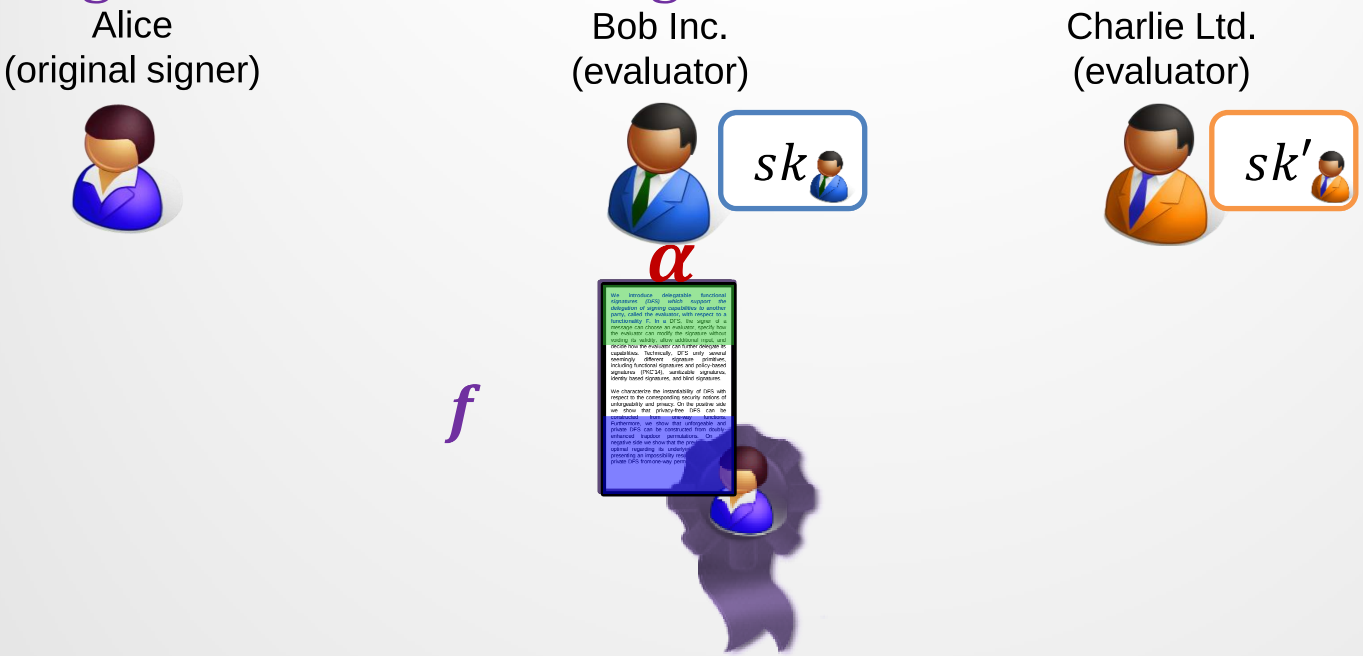


We introduce delegatable functional signatures (DFS) which support the delegation of signing capabilities to another party, called the evalua [illegible] with respect to a functionality F. In a DF [illegible] signer of a message can choose an [illegible] specify how the evaluator can [illegible] the signature without voiding its [illegible], allow additional input, and decide [illegible] the evaluator can further delegate its capabilities. Technically, DFS unify several seemingly different signature primitives, including functional signatures and policy-based signatures (PKC'14), sanitizable signatures, identity based signatures, and blind signatures.

We characterize the instantiability of DFS with respect to the corresponding security notions of unforgeability and privacy. On the positive side we show that privacy-free DFS can be constructed from one-way functions. Furthermore, we show that unforgeable and private DFS can be constructed from doubly-enhanced trapdoor permutations. On [illegible] negative side we show that the prev [illegible] optimal regarding its underlyi [illegible] presenting an impossibility resu [illegible] private DFS fromone-way perm [illegible]

- Alice signs a message and chooses how the message can be modified by which evaluator (Bob) and decides what Bob can further delegate, if at all.
- Bob modifies the message/signature pair, chooses how it can be further modified and by whom (Charlie).

Delegatable Functional Signatures

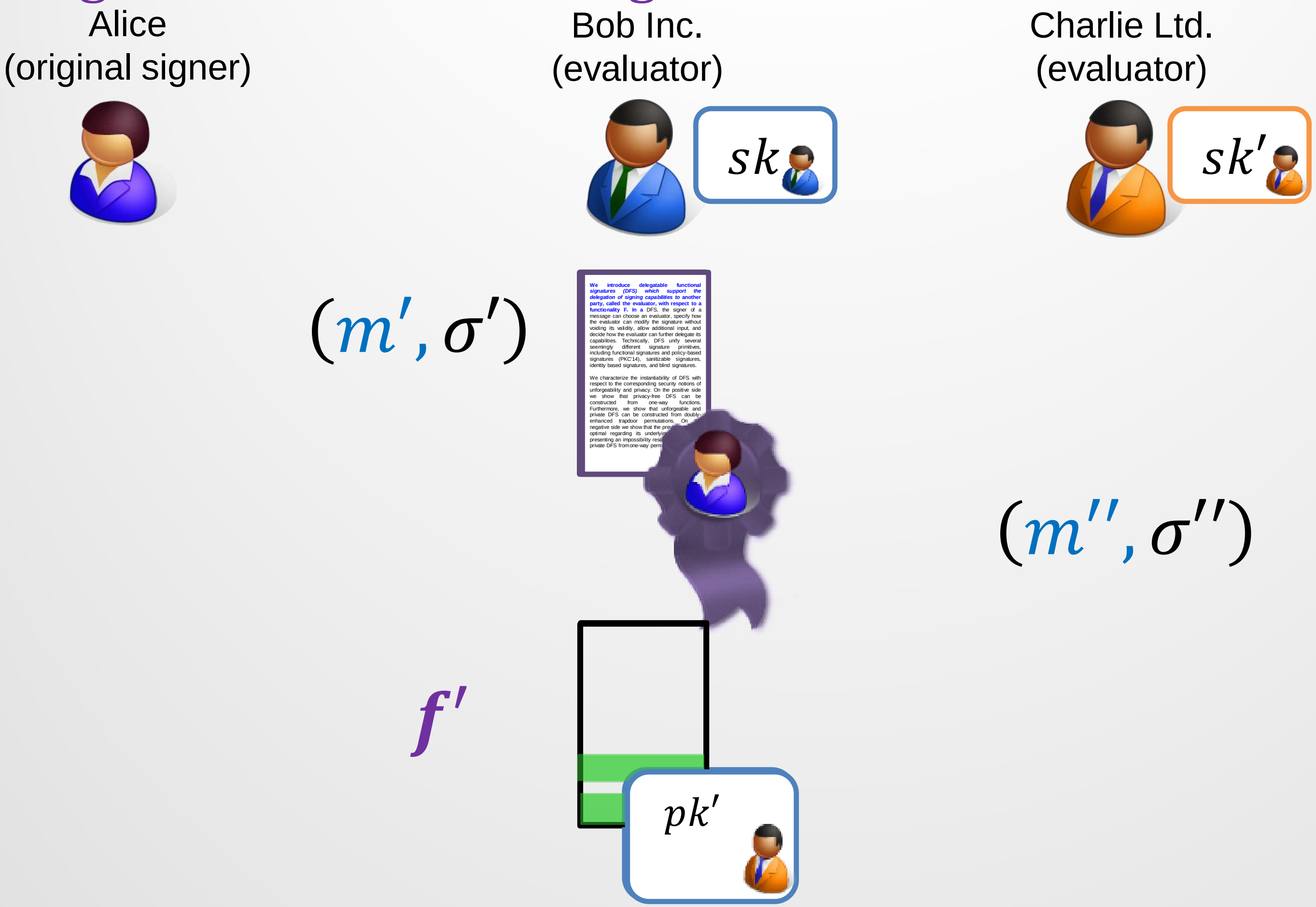


We introduce delegatable functional *signatures (DFS) which support the delegation of signing capabilities to another* party, called the evaluator, with respect to a functionality F. In a DFS, the signer of a message can choose an evaluator, specify how the evaluator can modify the signature without voiding its validity, allow additional input, and decide how the evaluator can further delegate its capabilities. Technically, DFS unify several seemingly different signature primitives, including functional signatures and policy-based signatures (PKC'14), sanitizable signatures, identity based signatures, and blind signatures.

We characterize the instantiability of DFS with respect to the corresponding security notions of unforgeability and privacy. On the positive side we show that privacy-free DFS can be constructed from one-way functions. Furthermore, we show that unforgeable and private DFS can be constructed from doubly-enhanced trapdoor permutations. On
negative side we show that the prev
optimal regarding its underlyin
presenting an impossibility resu
private DFS from one-way perm

- Alice signs a message and chooses how the message can be modified by which evaluator (Bob) and decides what Bob can further delegate, if at all.
- Bob modifies the message/signature pair, chooses how it can be further modified and by whom (Charlie).

Delegatable Functional Signatures



Alice
(original signer)

Bob Inc.
(evaluator)

sk

Charlie Ltd.
(evaluator)

sk'

(m', σ')

We introduce delegatable functional *signatures (DFS) which support the delegation of signing capabilities to* another party, called the evaluator, with respect to a functionality F. In a DFS, the signer of a message can choose an evaluator, specify how the evaluator can modify the signature without voiding its validity, allow additional input, and decide how the evaluator can further delegate its capabilities. Technically, DFS unify several seemingly different signature primitives, including functional signatures and policy-based signatures (PKC'14), sanitizable signatures, identity based signatures, and blind signatures.

We characterize the instantiability of DFS with respect to the corresponding security notions of unforgeability and privacy. On the positive side we show that privacy-free DFS can be constructed from one-way functions. Furthermore, we show that unforgeable and private DFS can be constructed from doubly-enhanced trapdoor permutations. On
negative side we show that the prev
optimal regarding its underlyin
presenting an impossibility resu
private DFS from one-way perm

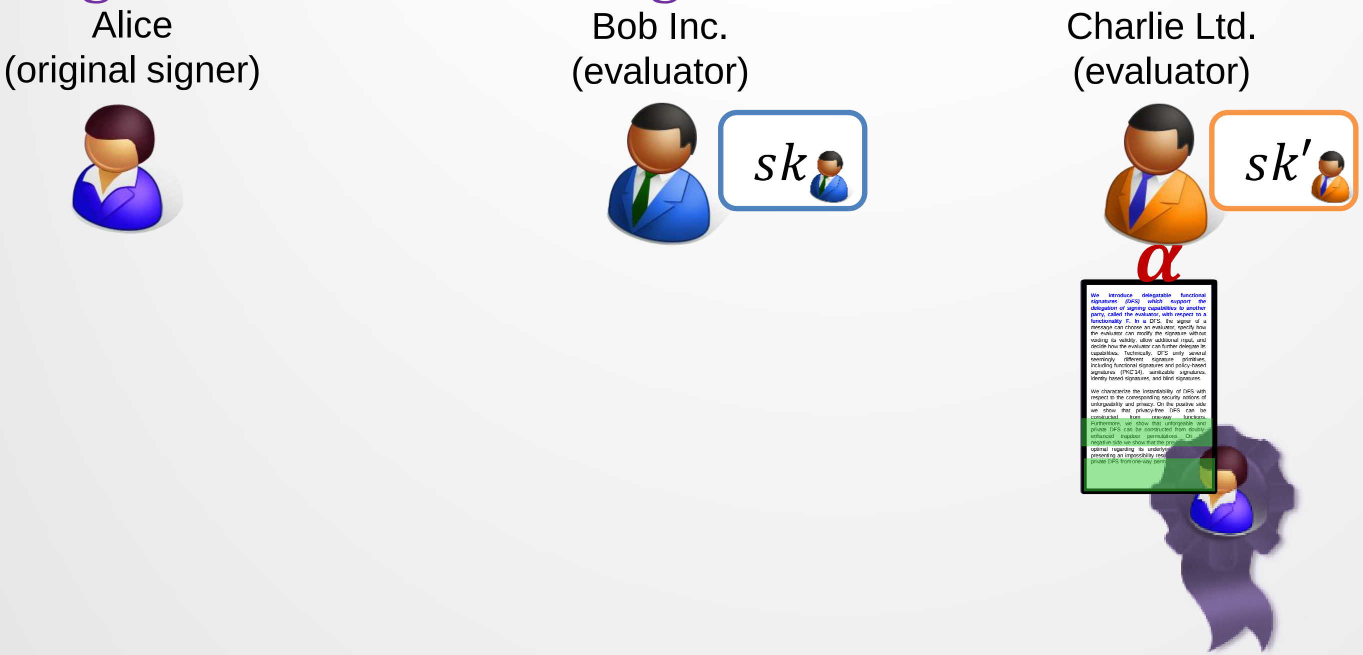
(m'', σ'')

f'

pk'

- Alice signs a message and chooses how the message can be modified by which evaluator (Bob) and decides what Bob can further delegate, if at all.
- Bob modifies the message/signature pair, chooses how it can be further modified and by whom (Charlie).

Delegatable Functional Signatures



Alice
(original signer)

Bob Inc.
(evaluator)

Charlie Ltd.
(evaluator)

We introduce delegatable functional *signatures (DFS) which support the delegation of signing capabilities to* another party, called the evaluator, with respect to a functionality F. In a DFS, the signer of a message can choose an evaluator, specify how the evaluator can modify the signature without voiding its validity, allow additional input, and decide how the evaluator can further delegate its capabilities. Technically, DFS unify several seemingly different signature primitives, including functional signatures and policy-based signatures (PKC'14), sanitizable signatures, identity based signatures, and blind signatures.

We characterize the instantiability of DFS with respect to the corresponding security notions of unforgeability and privacy. On the positive side we show that privacy-free DFS can be constructed from one-way functions. Furthermore, we show that unforgeable and private DFS can be constructed from doubly-enhanced trapdoor permutations. On
negative side we show that the pre
optimal regarding its underlyin
presenting an impossibility resu
private DFS from one-way perm

- Alice signs a message and chooses how the message can be modified by which evaluator (Bob) and decides what Bob can further delegate, if at all.
- Bob modifies the message/signature pair, chooses how it can be further modified and by whom (Charlie).

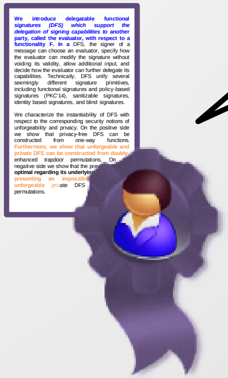
Delegatable Functional Signatures

Alice
(original signer)

Bob Inc.
(evaluator)

Charlie Ltd.
(evaluator)

Alice signed this message or allowed it!



We introduce delegatable functional *signatures (DFS) which support the delegation of signing capabilities to* another party, called the evaluator, with respect to a functionality F. In a DFS, the signer of a message can choose an evaluator, specify how the evaluator can modify the signature without voiding its validity, allow additional input, and decide how the evaluator can further delegate its capabilities. Technically, DFS unify several seemingly different signature primitives, including functional signatures and policy-based signatures (PKC'14), sanitizable signatures, identity based signatures, and blind signatures.

We characterize the instantiability of DFS with respect to the corresponding security notions of unforgeability and privacy. On the positive side we show that privacy-free DFS can be constructed from one-way functions. **Furthermore, we show that unforgeable and private DFS can be constructed from doubly-** enhanced trapdoor permutations. On negative side we show that the pre **optimal regarding its underlying** **presenting an impossibil** **unforgeable priv**ate DFS permutations.

$$(m'', \sigma'')$$

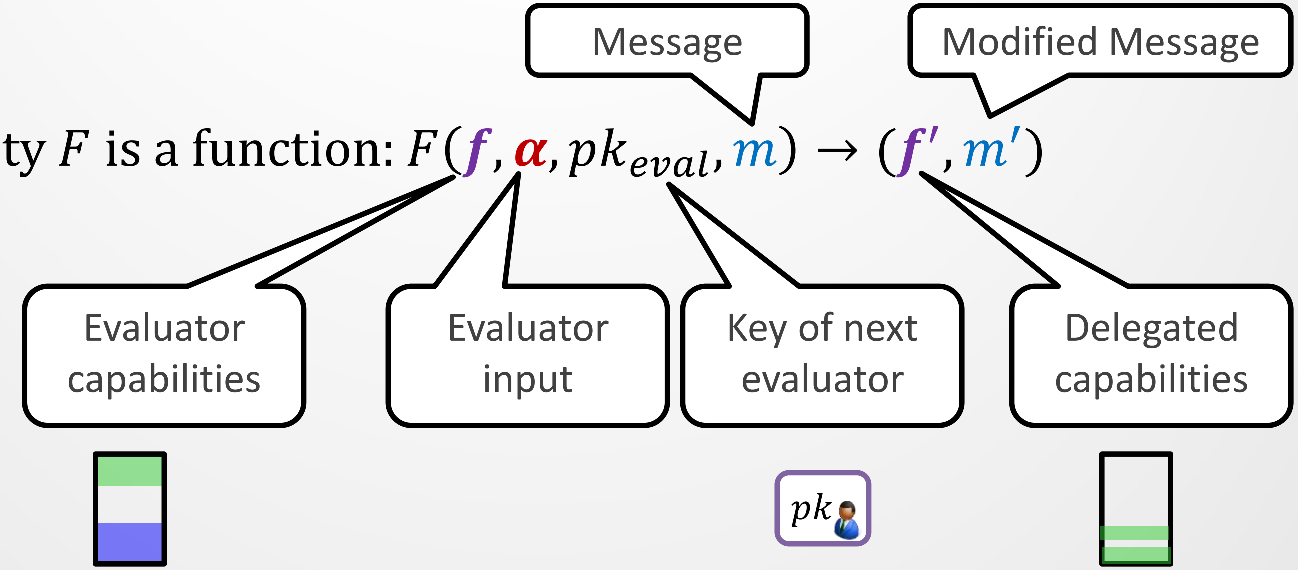
- Alice signs a message and chooses how the message can be modified by which evaluator (Bob) and decides what Bob can further delegate, if at all.
- Bob modifies the message/signature pair, chooses how it can be further modified and by whom (Charlie).

Overview

- Functionality and capabilities
- Security notions:
  - Types of adversaries
  - Unforgeability
  - Privacy
- Instantiability:
  - Privacy-free from one-way functions
  - Impossibility from one-way functions
  - Possibility from trapdoor permutations

Functionalities and their Transitive Closure

- A functionality F is a function: $F(\boldsymbol{f}, \boldsymbol{\alpha}, pk_{eval}, m) \to (\boldsymbol{f}', m')$



- Transitive Closure F^* for m and $\boldsymbol{f}$ with respect to the functionality F:
  - For $n = 0$: $F^0(\boldsymbol{f}, m) := \{(\boldsymbol{f}, m)\}$
  - For $n > 0$: $F^n(\boldsymbol{f}, m) := \{(\boldsymbol{f}, m)\} \cup_{\boldsymbol{\alpha}, pk_{eval}} F^{n-1}\big(\, F(\boldsymbol{f}, \boldsymbol{\alpha}, pk_{eval}, m)\big)$

$$F^*(\boldsymbol{f}, m) := \bigcup_{i=0}^{\infty} F^i(\boldsymbol{f}, m)$$

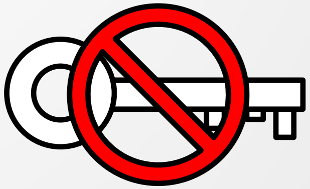
Overview

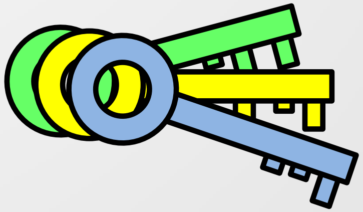
- Functionality and capabilities
- Security notions:
  - Types of adversaries
  - Unforgeability
  - Privacy
- Instantiability:
  - Privacy-free from one-way functions
  - Impossibility from one-way functions
  - Possibility from trapdoor permutations

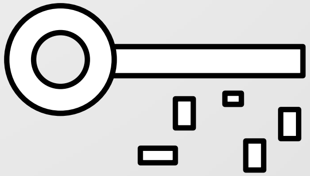
Security Notions – Adversaries

- Three different types of adversaries:
  - Outsider:
    - Access to an oracle for public evaluator keys.
    - No access to secret evaluator keys.
  - Insider:
    - Access to an oracle for public evaluator keys.
    - Access to an oracle for secret evaluator keys.
  - Strong Insider:
    - Access to an oracle for public evaluator keys.
    - Access to an oracle for secret evaluator keys.
    - Can register its own secret evaluator keys.

Secret Evaluator Key(s):







Unforgeability – Intuition

- The adversary can request message/signature pairs; fresh ones as well as modified ones.

$$(m, \sigma) \rightarrow (m', \sigma')$$

- The adversary should not be able to generate valid (verifying) message/signature pairs that are not allowed by the signer.

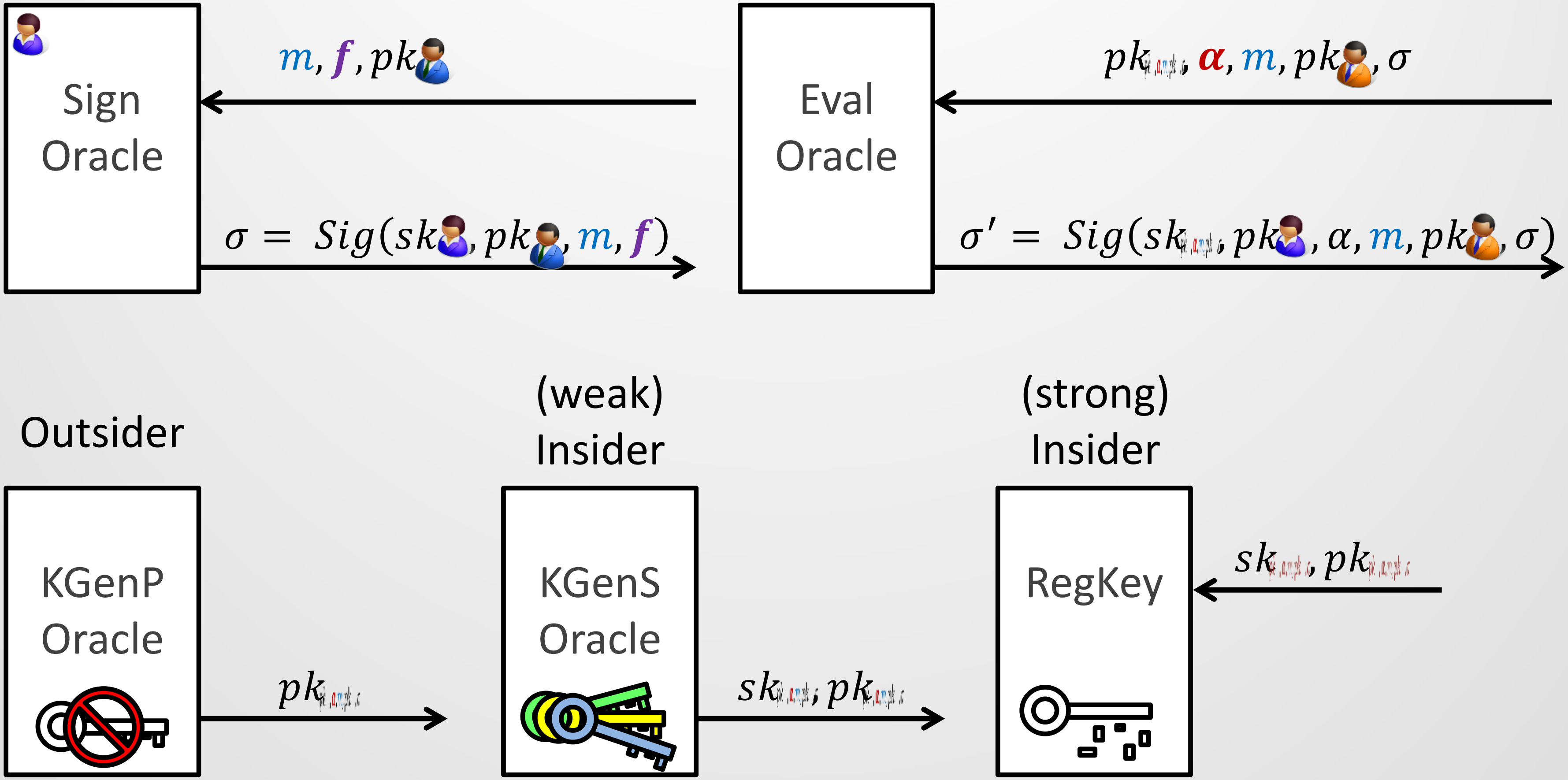
$$(m^*, \sigma^*)$$

- All “forgeries” that were allowed by the signer, modified by legitimate evaluators or by the adversary (if delegated to it) are discarded.

$\forall (m, \boldsymbol{f})\ of$ 👤 $, \forall \boldsymbol{f}.$

$$(m^*, \boldsymbol{f}) \notin F^*(\boldsymbol{f}, m)$$

Unforgeability – Oracles



Sign Oracle

m, f, pk

$\sigma = Sig(sk, pk, m, f)$

Eval Oracle

$pk, \alpha, m, pk, \sigma$

$\sigma' = Sig(sk, pk, \alpha, m, pk, \sigma)$

Outsider

KGenP Oracle

pk

(weak) Insider

KGenS Oracle

sk, pk

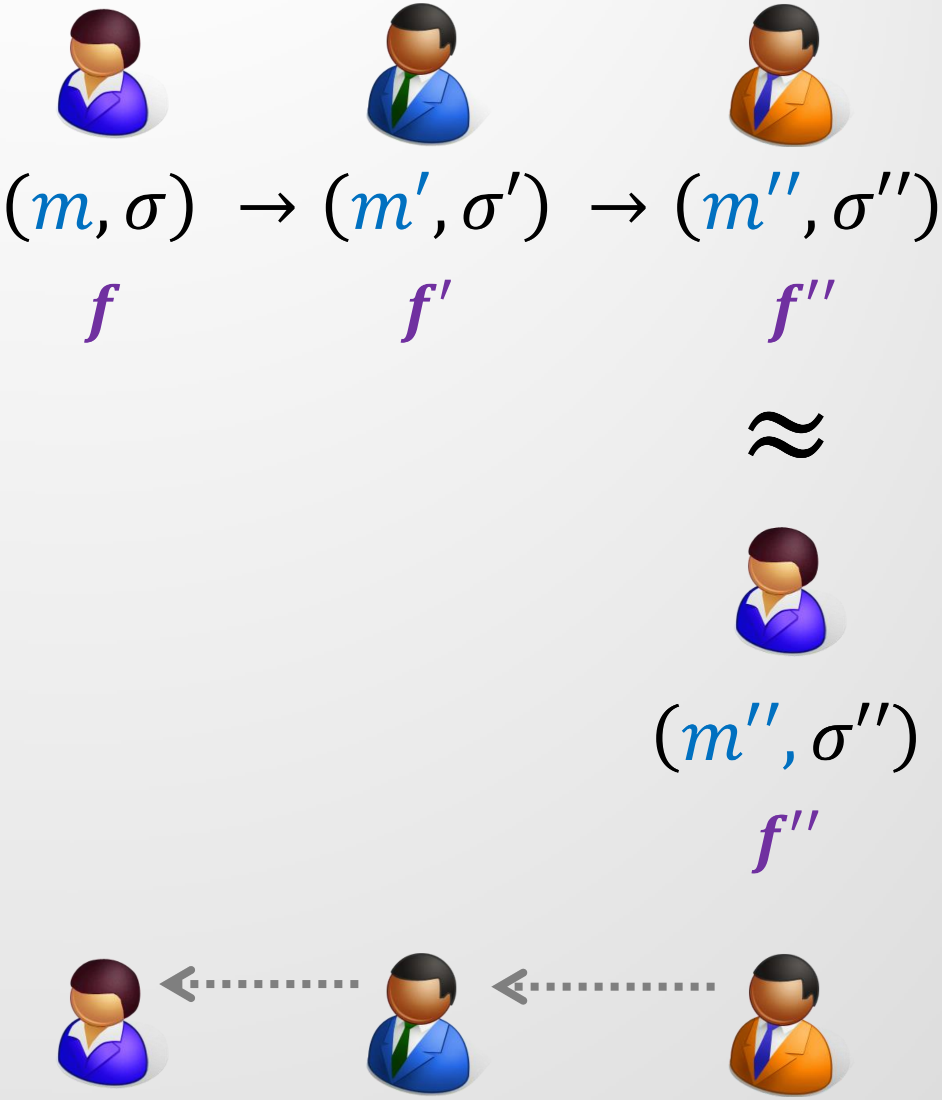
(strong) Insider

RegKey

sk, pk

Privacy (under Chosen Function Attacks) – Intuition

- The adversary should be unable to distinguish a signature that has been modified from a fresh signature for the same message.
- Conditions and Exceptions:
  - The message (m'') has to be the same.
  - The capabilities (f'') have to be the same.
  - Each evaluator may learn something about the previous party in the line (for verifying the previous step).



$$(m,\sigma) \rightarrow (m',\sigma') \rightarrow (m'',\sigma'')$$

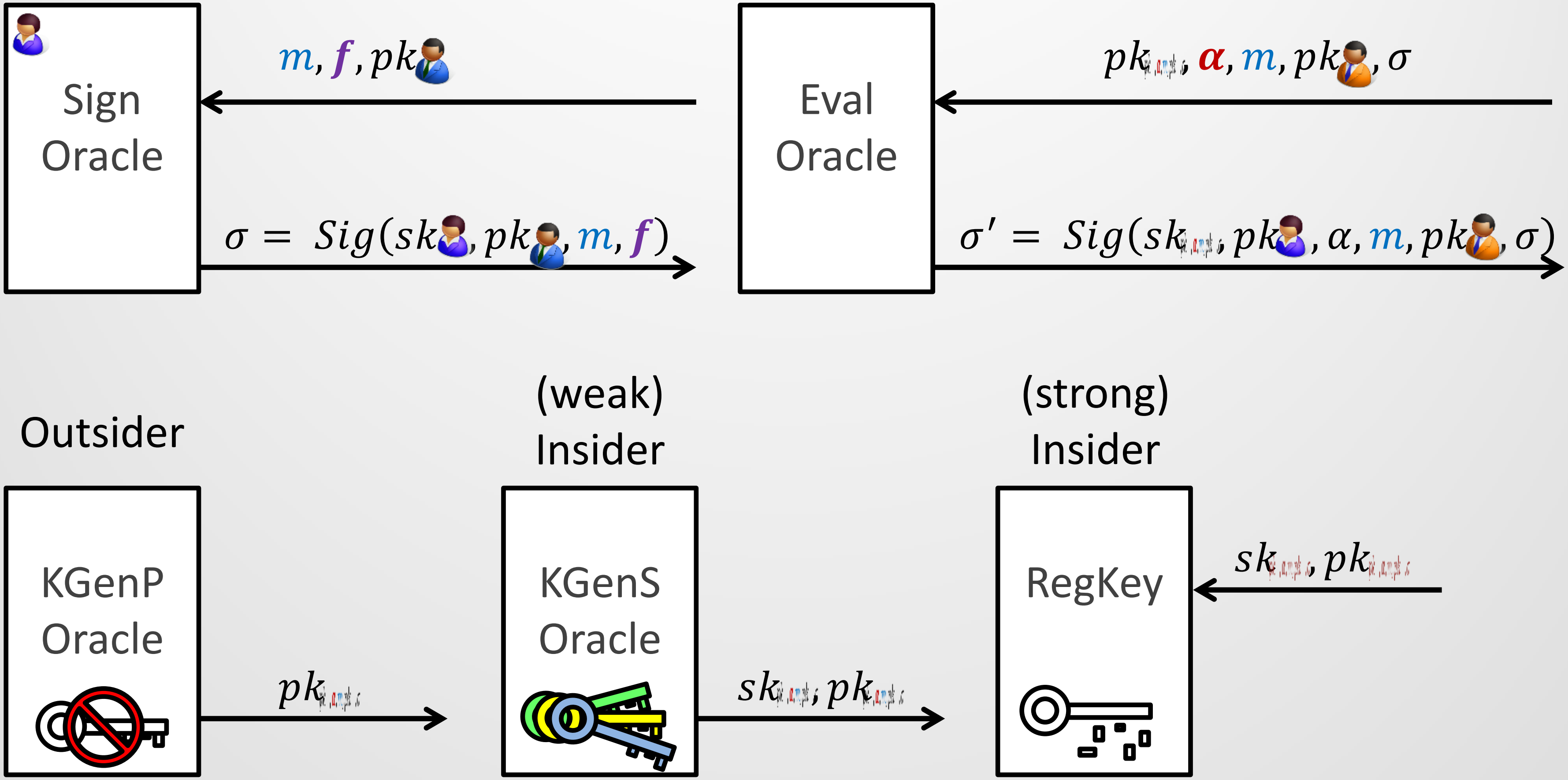
f f' f''

$$\approx$$

$$(m'',\sigma'')$$

f''

Privacy – Reminder of the Oracles



Sign Oracle

m, f, pk

$\sigma = Sig(sk, pk, m, f)$

Eval Oracle

$pk, \alpha, m, pk, \sigma$

$\sigma' = Sig(sk, pk, \alpha, m, pk, \sigma)$

Outsider

KGenP Oracle

pk

(weak) Insider

KGenS Oracle

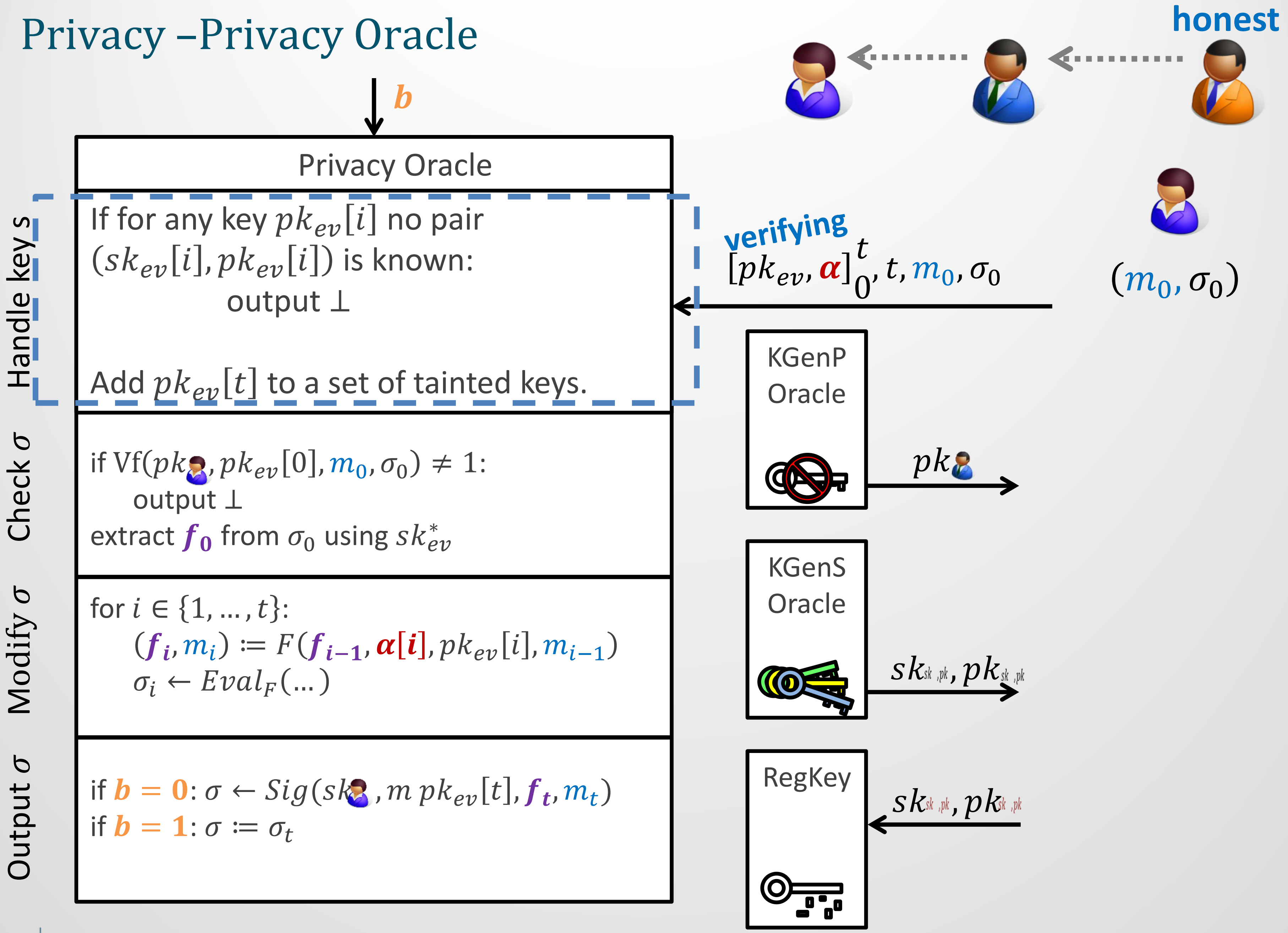
sk, pk

(strong) Insider

RegKey

sk, pk

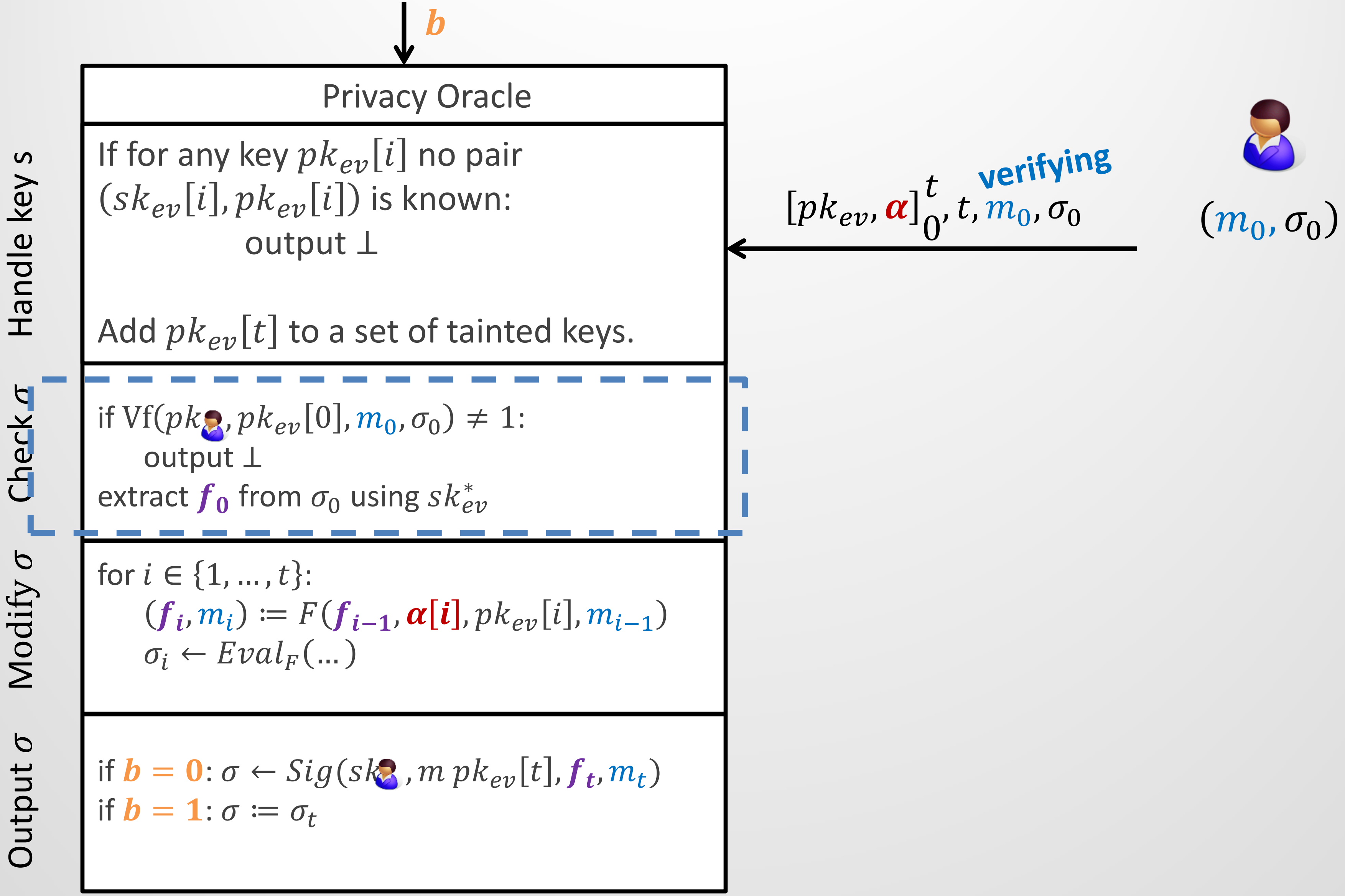
Privacy –Privacy Oracle



Privacy Oracle

```
If for any key $pk_{ev}[i]$ no pair
$(sk_{ev}[i], pk_{ev}[i])$ is known:
        output $\perp$

Add $pk_{ev}[t]$ to a set of tainted keys.

if $\mathrm{Vf}(pk, pk_{ev}[0], m_0, \sigma_0) \neq 1$:
    output $\perp$
extract $f_0$ from $\sigma_0$ using $sk_{ev}^*$

for $i \in \{1, \ldots, t\}$:
    $(f_i, m_i) \coloneqq F(f_{i-1}, \alpha[i], pk_{ev}[i], m_{i-1})$
    $\sigma_i \leftarrow Eval_F(\ldots)$

if $b = 0$: $\sigma \leftarrow Sig(sk, m\ pk_{ev}[t], f_t, m_t)$
if $b = 1$: $\sigma \coloneqq \sigma_t$
```

Privacy –Privacy Oracle



b

Privacy Oracle

Handle key s

```
If for any key $pk_{ev}[i]$ no pair
$(sk_{ev}[i], pk_{ev}[i])$ is known:
        output $\perp$

Add $pk_{ev}[t]$ to a set of tainted keys.
```

Check σ

```
if Vf$(pk_{👤}, pk_{ev}[0], m_0, \sigma_0) \neq 1$:
    output $\perp$
extract $\boldsymbol{f_0}$ from $\sigma_0$ using $sk_{ev}^*$
```

Modify σ

```
for $i \in \{1, \ldots, t\}$:
    $(\boldsymbol{f_i}, m_i) \coloneqq F(\boldsymbol{f_{i-1}}, \boldsymbol{\alpha}[\boldsymbol{i}], pk_{ev}[i], m_{i-1})$
    $\sigma_i \leftarrow Eval_F(\ldots)$
```

Output σ

```
if $\boldsymbol{b} = \boldsymbol{0}$: $\sigma \leftarrow Sig(sk_{👤}, m\ pk_{ev}[t], \boldsymbol{f_t}, m_t)$
if $\boldsymbol{b} = \boldsymbol{1}$: $\sigma \coloneqq \sigma_t$
```

verifying

$[pk_{ev}, \boldsymbol{\alpha}]_0^t, t, m_0, \sigma_0$

(m_0, σ_0)

Privacy –Privacy Oracle

b

Privacy Oracle

Handle key s

```
If for any key $pk_{ev}[i]$ no pair
$(sk_{ev}[i], pk_{ev}[i])$ is known:
        output $\bot$

Add $pk_{ev}[t]$ to a set of tainted keys.
```

Check σ

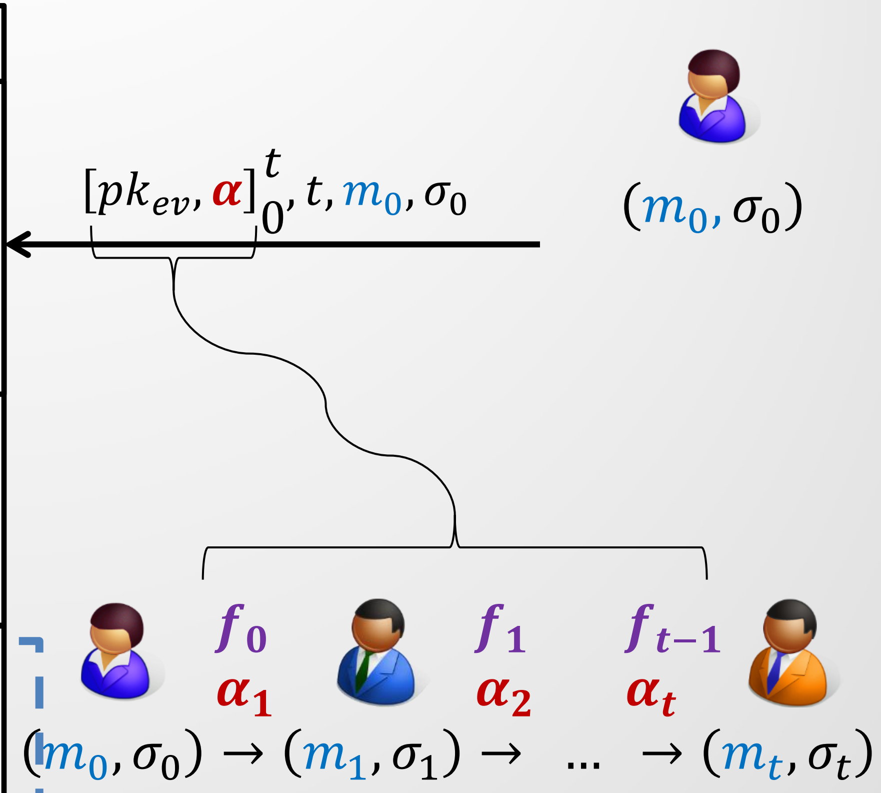
```
if Vf$(pk, pk_{ev}[0], m_0, \sigma_0) \neq 1$:
    output $\bot$
extract $f_0$ from $\sigma_0$ using $sk^*_{ev}$
```

Modify σ

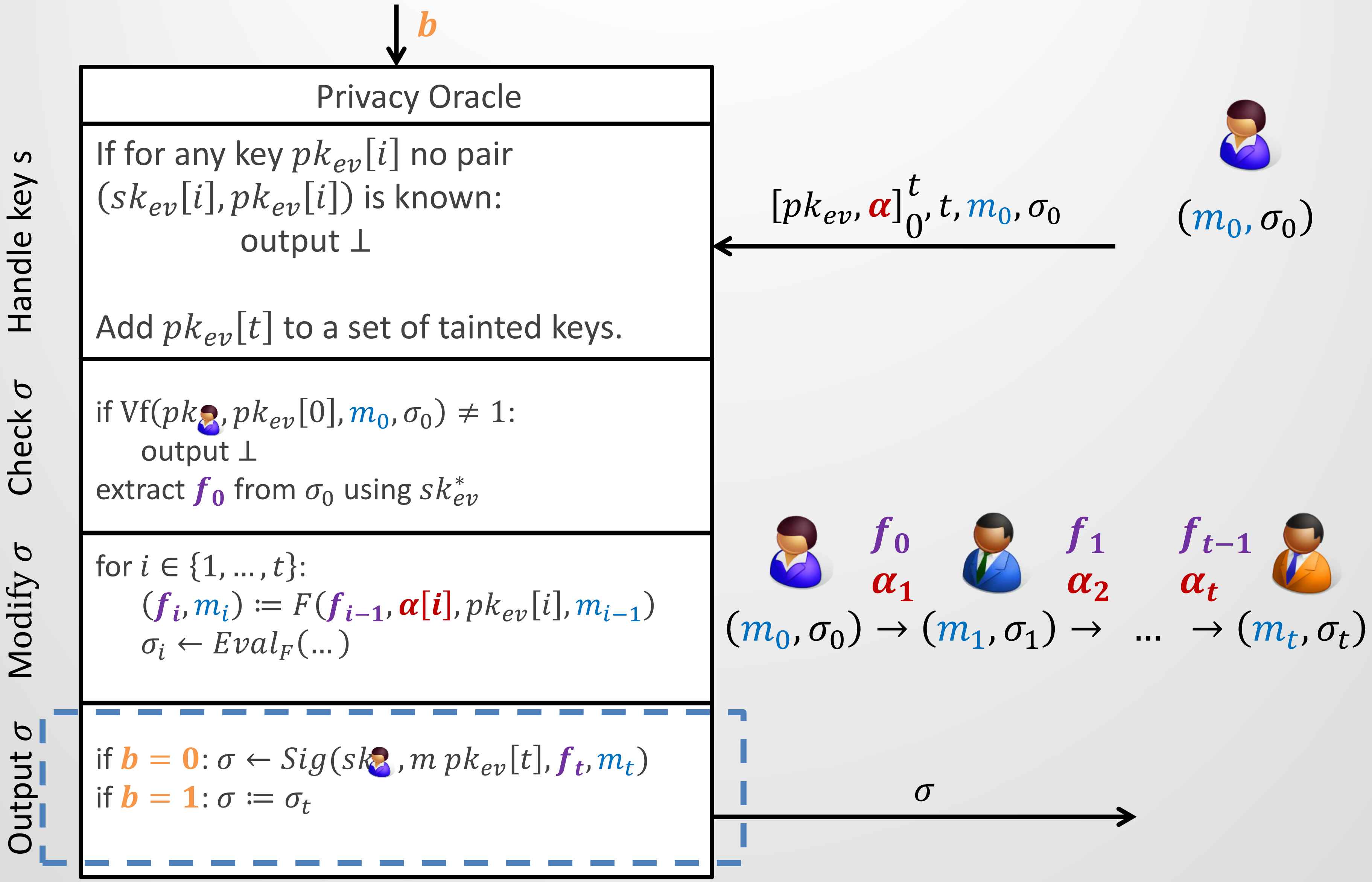
```
for $i \in \{1, \dots, t\}$:
    $(f_i, m_i) := F(f_{i-1}, \alpha[i], pk_{ev}[i], m_{i-1})$
    $\sigma_i \leftarrow Eval_F(\dots)$
```

Output σ

```
if $b = 0$: $\sigma \leftarrow Sig(sk, m\ pk_{ev}[t], f_t, m_t)$
if $b = 1$: $\sigma := \sigma_t$
```


$[pk_{ev}, \alpha]_0^t, t, m_0, \sigma_0$
(m_0, σ_0)
f_0
α_1
f_1
α_2
f_{t-1}
α_t
$(m_0, \sigma_0) \rightarrow (m_1, \sigma_1) \rightarrow \dots \rightarrow (m_t, \sigma_t)$

Privacy –Privacy Oracle



b

Privacy Oracle

Handle keys

```
If for any key $pk_{ev}[i]$ no pair
$(sk_{ev}[i], pk_{ev}[i])$ is known:
        output ⊥

Add $pk_{ev}[t]$ to a set of tainted keys.
```

Check σ

```
if $\mathrm{Vf}(pk, pk_{ev}[0], m_0, \sigma_0) \neq 1$:
    output ⊥
extract $f_0$ from $\sigma_0$ using $sk_{ev}^*$
```

Modify σ

```
for $i \in \{1, \dots, t\}$:
    $(f_i, m_i) \coloneqq F(f_{i-1}, \alpha[i], pk_{ev}[i], m_{i-1})$
    $\sigma_i \leftarrow Eval_F(\dots)$
```

Output σ

```
if $b = 0$: $\sigma \leftarrow Sig(sk, m\ pk_{ev}[t], f_t, m_t)$
if $b = 1$: $\sigma \coloneqq \sigma_t$
```

$[pk_{ev}, \alpha]_0^t, t, m_0, \sigma_0$

(m_0, σ_0)

f_0 α_1 f_1 α_2 f_{t-1} α_t

$(m_0, \sigma_0) \to (m_1, \sigma_1) \to \dots \to (m_t, \sigma_t)$

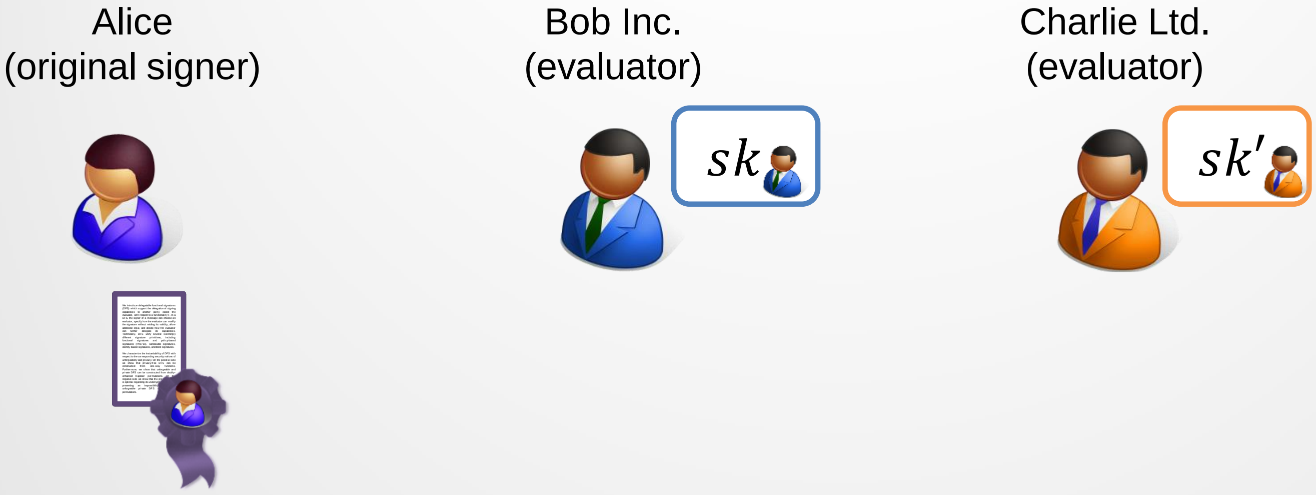
σ

Overview

- Functionality and capabilities
- Security notions:
  - Types of adversaries
  - Unforgeability
  - Privacy
- Instantiability:
  - Privacy-free from one-way functions
  - Impossibility from one-way functions
  - Possibility from trapdoor permutations

Instantiation from OWF (without Privacy)

Requires:
one-way functions



Alice
(original signer)

Bob Inc.
(evaluator)

sk

Charlie Ltd.
(evaluator)

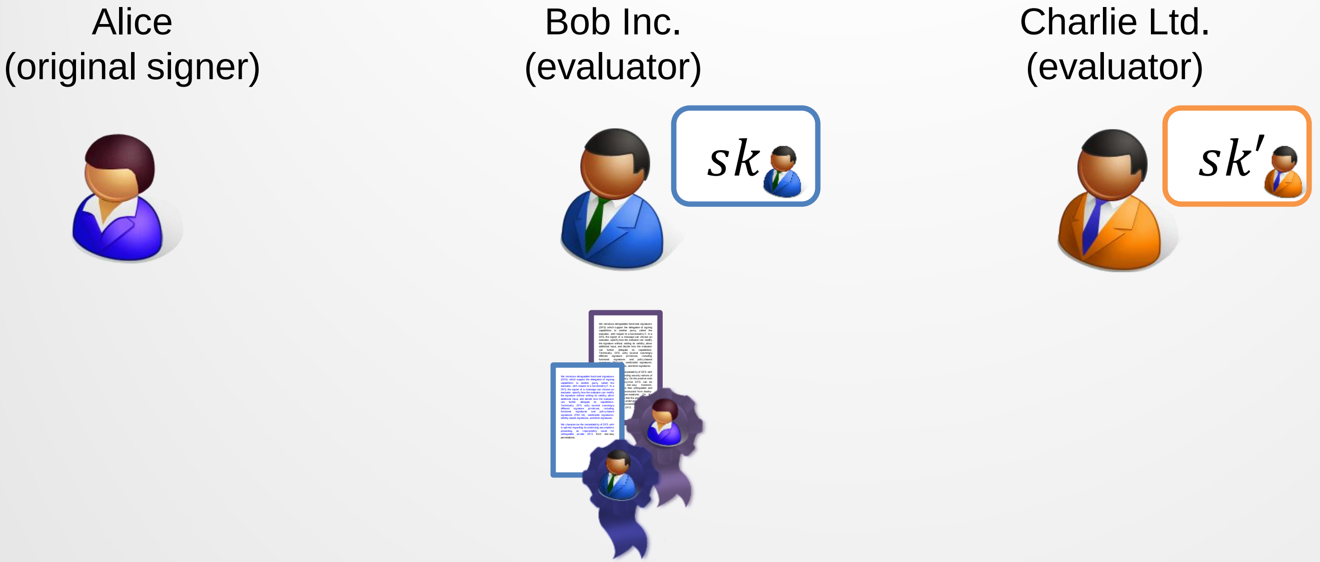
sk'

Idea: authentication chain

- Alice signs a message and a functionality with her secret key.
- Bob appends his changes and signs them (and the message/signature upon which they are based) with his secret key.
- Charlie appends his changes and signs them (and the message/signature upon which they are based) with his secret key.

Instantiation from OWF (without Privacy)

Requires:
one-way functions



Idea: authentication chain

- Alice signs a message and a functionality with her secret key.
- Bob appends his changes and signs them (and the message/signature upon which they are based) with his secret key.
- Charlie appends his changes and signs them (and the message/signature upon which they are based) with his secret key.

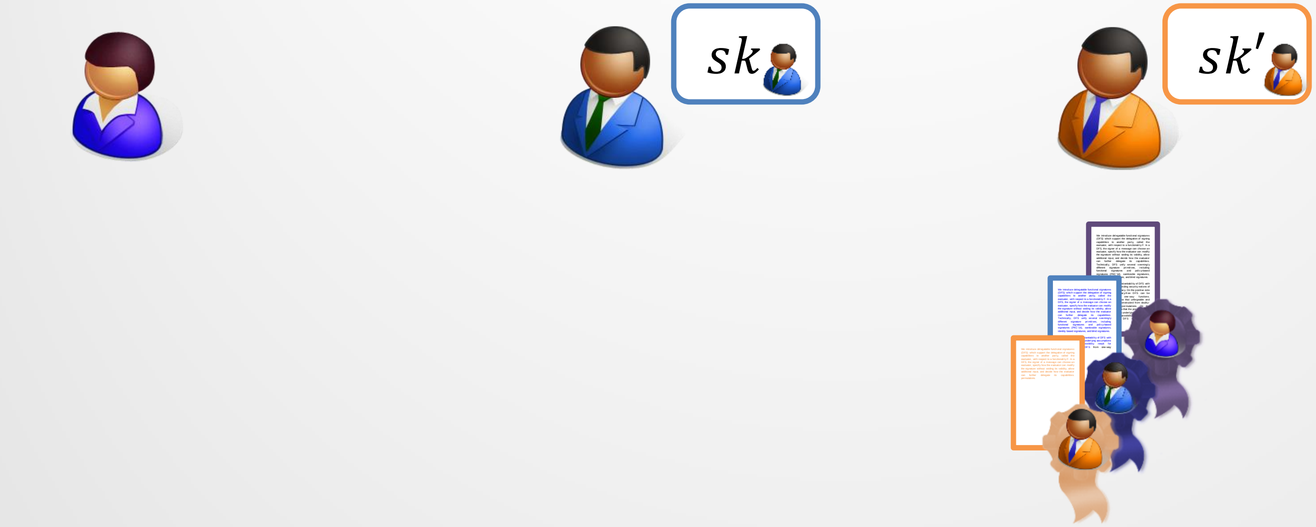
Instantiation from OWF (without Privacy)

Requires:
one-way functions

Alice
(original signer)

Bob Inc.
(evaluator)

Charlie Ltd.
(evaluator)



Idea: authentication chain

- Alice signs a message and a functionality with her secret key.
- Bob appends his changes and signs them (and the message/signature upon which they are based) with his secret key.
- Charlie appends his changes and signs them (and the message/signature upon which they are based) with his secret key.

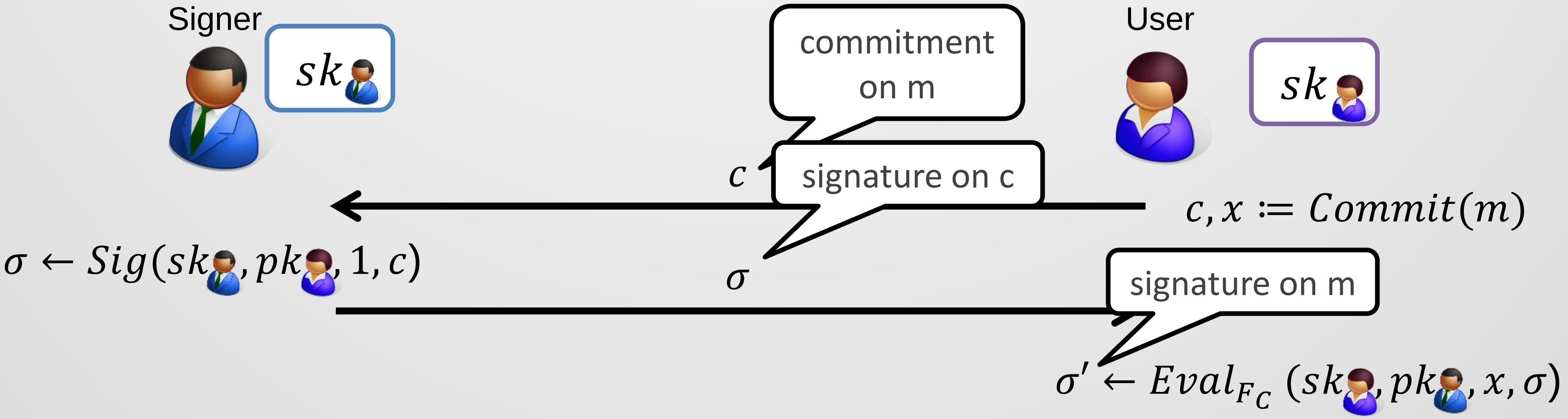
Impossibility with Privacy

Construction from
one-way permutations
is impossible.

- Idea: We construct **blind signatures** from DFS using black-box techniques.
- **Blind signatures** cannot be constructed from one-way permutations using black-box techniques [KSY – TCC’11].
- Functionality:

$$F_C(\mathbf{1}, \boldsymbol{\alpha}, pk_{user}, m) \coloneqq (\mathbf{0}, Open(\boldsymbol{\alpha}, m))$$

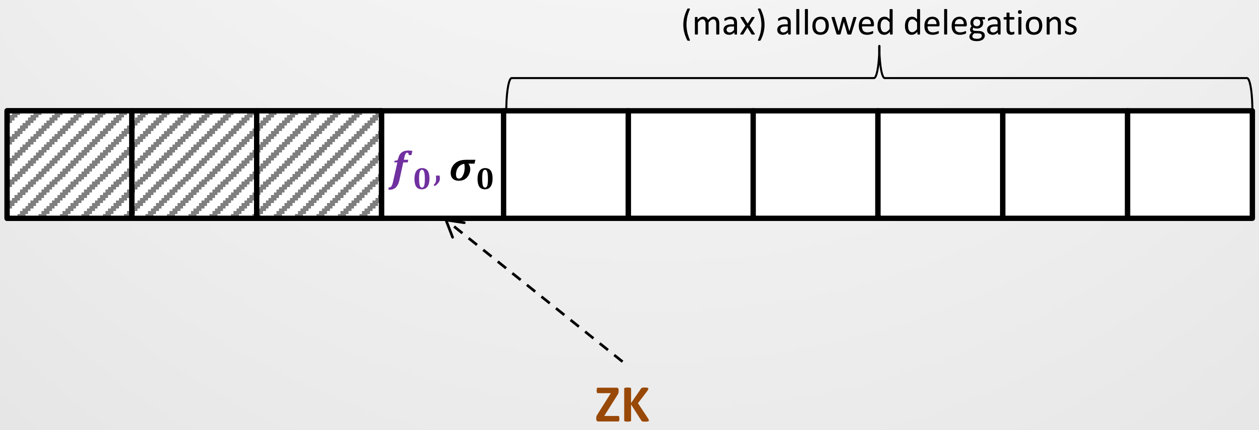
$$(Commit(m), \sigma_0) \rightarrow (m, \sigma_1)$$


Signer
sk
User
sk
commitment on m
c
signature on c
$c, x \coloneqq Commit(m)$
$\sigma \leftarrow Sig(sk, pk, 1, c)$
σ
signature on m
$\sigma' \leftarrow Eval_{F_C}(sk, pk, x, \sigma)$

Instantiation from trapdoor permutations

Construction from trapdoor permutations.

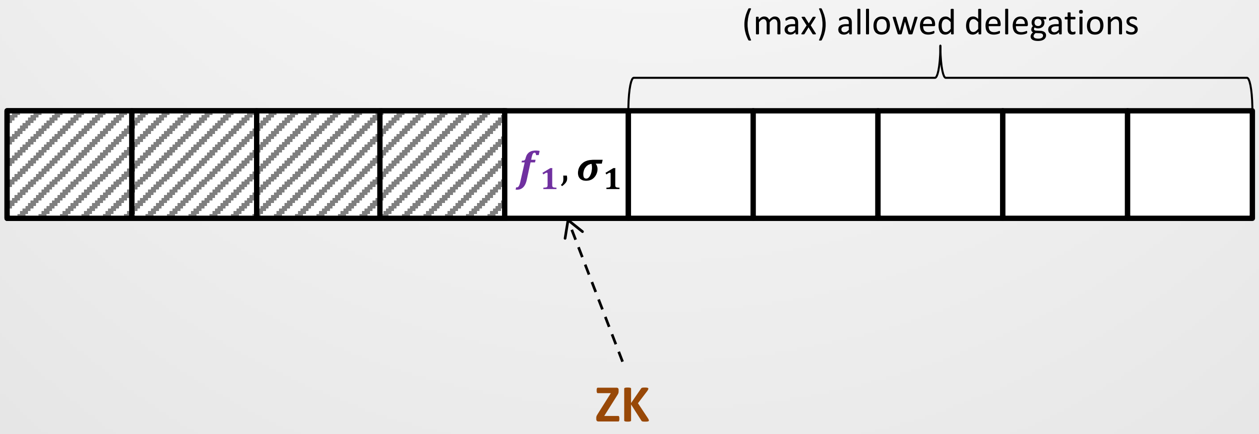
- Idea: Encrypt and prove.
  - Each evaluator verifies the signature of the previous party.
  - Encrypt the transcript of all signatures (pre-allocate enough space).
  - Zero Knowledge proofs that the signature chain is valid.



Instantiation from trapdoor permutations

Construction from trapdoor permutations.

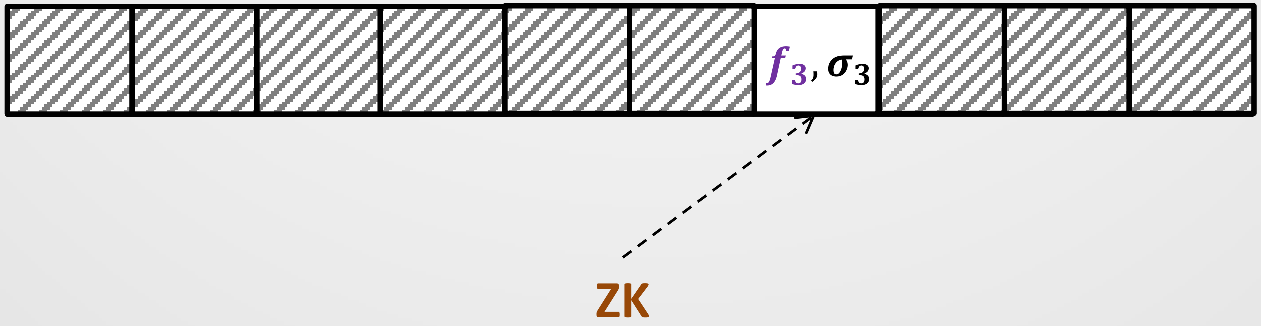
- Idea: Encrypt and prove.
  - Each evaluator verifies the signature of the previous party.
  - Encrypt the transcript of all signatures (pre-allocate enough space).
  - Zero Knowledge proofs that the signature chain is valid.



Instantiation from trapdoor permutations

Construction from trapdoor permutations.

- Idea: Encrypt and prove.
  - Each evaluator verifies the signature of the previous party.
  - Encrypt the transcript of all signatures (pre-allocate enough space).
  - Zero Knowledge proofs that the signature chain is valid.



Open Problems

- Construction for unbounded number of delegations
- Efficient Construction
- Signatures with constant size

Open Problems

Thank you
for your attention!

Questions?