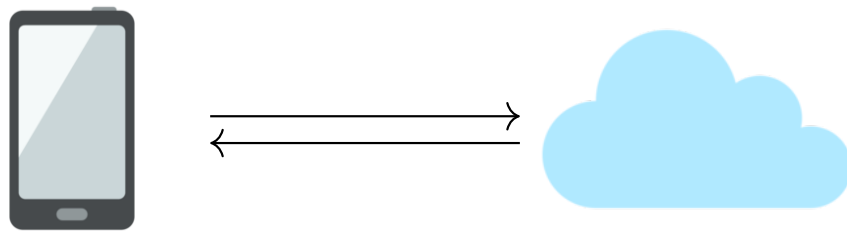
Steel: Composable Hardware-based Stateful and Randomised Functional Encryption

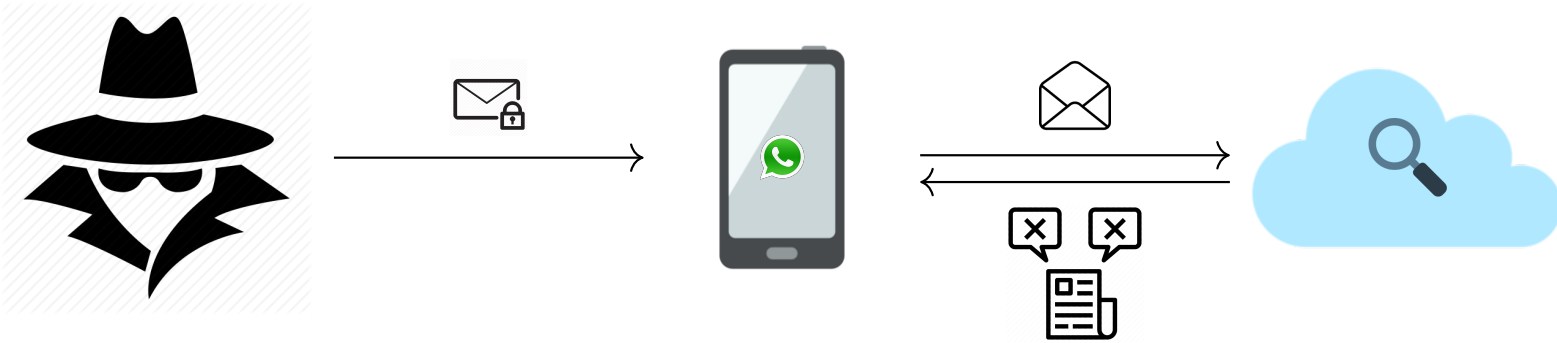
PKC 2021

Pramod Bhatotia, Markulf Kohlweiss, Lorenzo Martinico, Yiannis Tselekounis

Motivation



Motivation



Public Key Functional Encryption

Syntax

- $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{Init}$: one-time setup
- $\mathsf{sk}_\mathsf{F} \leftarrow \mathit{Keygen}(\mathsf{msk}, \mathsf{F})$: produces a functional key
- $\mathsf{ct} \leftarrow \mathit{Enc}(\mathsf{mpk}, \mathsf{x})$
- $\mathsf{F}(\mathsf{m}) \leftarrow \mathit{Dec}(\mathsf{sk}_\mathsf{F}, \mathsf{ct})$: evaluates function

Properties

- Authorises the decryption of a function evaluation
- Secret input, public output

Public Key Functional Encryption

Charlie

Alice

Bob

Public Key Functional Encryption

Charlie

Alice

mpk

Init()

Bob

Public Key Functional Encryption

Charlie

Alice

F?

Bob

Public Key Functional Encryption

Charlie

Alice

sk_F

Bob

Public Key Functional Encryption

Charlie

Alice

sk_F

Bob

ct

$ct \leftarrow Enc(mpk, x)$

Public Key Functional Encryption

Charlie

Alice

sk_F

Bob

ct

$ct \leftarrow Enc(mpk, x)$

$Dec(sk_F, ct)$

$F(x)$

Current Limitations

- Non-trivial to construct in practice
- Efficient realisations for limited class of functions (deterministic, dot-product)
- Composability is impossible in the standard model (Matt, Maurer 2015)

Hardware-based solutions

Trusted Execution Environments

Hardware-based FE

- Iron (Fisch et al, 2016) realises Functional Encryption using Trusted hardware

Contributions

- Introduce Stateful & Randomised Functional encryption (FESR)
- Extend Iron and formalise security under the UC model of Pass et al, 2017 (PST in short)
- Relax the PST model to capture additional adversaries

Our protocol Steel:

- composable
- hardware-based
- stateful and randomised functional encryption

FESR functionality

FESR

$s_0 = \emptyset$

FESR functionality

F

FESR

$s_0 = \emptyset$

FESR functionality

F

x_0

FESR

$s_0 = \emptyset$

FESR functionality

F

x_0

FESR

$(y_0, s_1) \xleftarrow{r} F(x_0, s_0)$

y_0

FESR functionality

F

x_1

FESR

s_1

FESR functionality

F

x_1

FESR

$(y_1, s_2) \xleftarrow{r} F(x_1, s_1)$

y_1

Properties

Confidentiality

Bob learns only the output of authorised functions
$(y, s') \leftarrow F(x, s; r)$

Correctness

The state is determined by the sequence of decryptions (for each unique function and decryptor)
$(y_n, s_{n+1}) \leftarrow F(x_n, s_n; r_n) \ldots (y_0, s_1) \leftarrow F(x_0, \emptyset; r_0)$

TEE architecture

OS

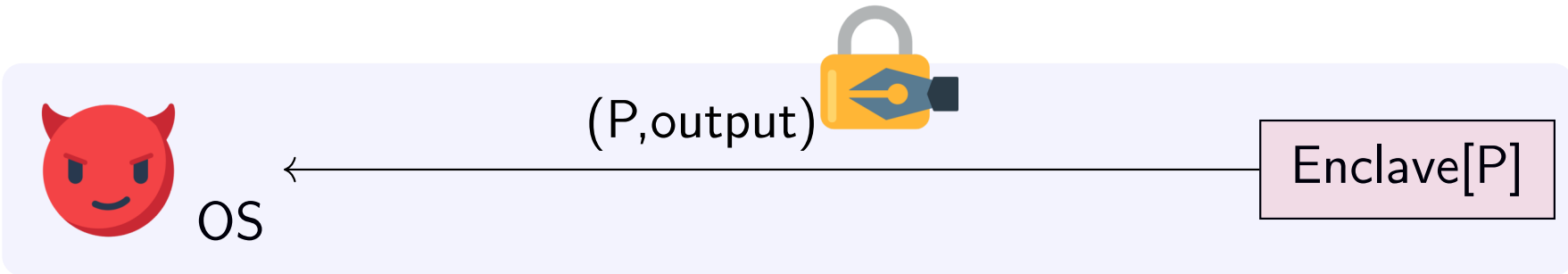
Enclave[P]

- Allows running sensitive data on untrusted host

TEE architecture

- Allows running sensitive data on untrusted host
- Protect code and data (confidentiality)

TEE architecture



- Allows running sensitive data on untrusted host
- Protect code and data (confidentiality)
- Attest to return value (integrity)

PST Model

- Abstracts TEEs as a UC global functionality G_{att}

Interface

- $(\text{mpk}, \text{msk}) \leftarrow \mathit{Init}$: one-time only, sets up signature scheme parameters
- $\text{mpk} \leftarrow \mathit{GetPK}$: executable by any party with no access to a TEE
- $\text{eid} \leftarrow \mathit{Install}(\mathit{prog})$: install an enclave on a particular machine
- $(\mathit{out}, \sigma) \leftarrow \mathit{Resume}(\text{eid}, \mathit{input})$: executes $\mathit{prog}(\mathit{input})$ and returns attested output
  - $\sigma = \Sigma.\mathit{sign}(\text{msk}, (\text{eid}, \mathit{prog}, \mathit{out}))$

Iron and Steel principles

- Bob and Charlie are SGX-equipped
- Encryption is just plain PKE Encryption
- Key material is kept within enclave and exchanged through attestation
- Functional keys are signatures over a function representation

Steel protocol

crs

Key Mgmt

crs

crs

Dec

Steel protocol

crs, mpk $\longleftarrow$ **Key Mgmt**
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs

crs

Dec

Steel protocol

crs, mpk

Key Mgmt

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs,mpk

crs

Dec

Steel protocol

crs, mpk

Key Mgmt

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs,mpk

crs

Dec

Steel protocol

crs, mpk

Key Mgmt

$(\text{mpk}, \text{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs,mpk

(mpk, msk)

crs

Dec

Steel protocol

crs, mpk

Key Mgmt

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs,mpk

F?

crs

Dec

Steel protocol

crs, mpk

Key Mgmt

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs,mpk

$\mathsf{sk}_F = (F, \text{🔒})$

crs

Dec

Steel protocol

crs, mpk

Key Mgmt

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs,mpk

$\mathsf{ct} \xleftarrow{r} \mathit{Enc}(\mathsf{mpk}, x)$

$\pi \leftarrow \mathcal{P}((\mathsf{mpk}, \mathsf{ct}), (x, r), \mathsf{crs})$

crs

Dec

Steel protocol

crs, mpk

Key Mgmt

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs,mpk

$\mathsf{ct} \xleftarrow{r} \mathit{Enc}(\mathsf{mpk}, x)$

$\pi \leftarrow \mathcal{P}((\mathsf{mpk}, \mathsf{ct}), (x, r), \mathsf{crs})$

crs, (ct, π)

Func$[F]$

$s \leftarrow \emptyset$

Dec

Steel protocol

crs, mpk

Key Mgmt

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs,mpk

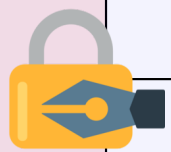
$\mathsf{ct} \xleftarrow{r} \mathit{Enc}(\mathsf{mpk}, x)$

$\pi \leftarrow \mathcal{P}((\mathsf{mpk}, \mathsf{ct}), (x, r), \mathsf{crs})$

crs, (ct, π)

Func$[F]$

$s \leftarrow \emptyset$



Dec

Steel protocol

crs, mpk

Key Mgmt
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs,mpk
$\mathsf{ct} \xleftarrow{r} \mathit{Enc}(\mathsf{mpk}, x)$
$\pi \leftarrow \mathcal{P}((\mathsf{mpk}, \mathsf{ct}), (x, r), \mathsf{crs})$

$\mathsf{crs}, (\mathsf{ct}, \pi)$

Func[F]

$s \leftarrow \emptyset$

msk

Dec
if F is authorised

Steel protocol

crs, mpk

Key Mgmt
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathit{PKE.KeyGen}()$

crs,mpk
$\mathsf{ct} \xleftarrow{r} \mathit{Enc}(\mathsf{mpk}, x)$
$\pi \leftarrow \mathcal{P}((\mathsf{mpk}, \mathsf{ct}), (x, r), \mathsf{crs})$

crs, (ct, π)

Func[F]
$\mathcal{V}((\mathsf{mpk}, \mathsf{ct}), \pi, \mathsf{crs})$
$s \leftarrow \emptyset$

Dec

Steel protocol

crs, mpk

Key Mgmt

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow PKE.KeyGen()$

crs,mpk

$\mathsf{ct} \xleftarrow{r} Enc(\mathsf{mpk}, x)$

$\pi \leftarrow \mathcal{P}((\mathsf{mpk}, \mathsf{ct}), (x, r), \mathsf{crs})$

crs, y

Func[F]

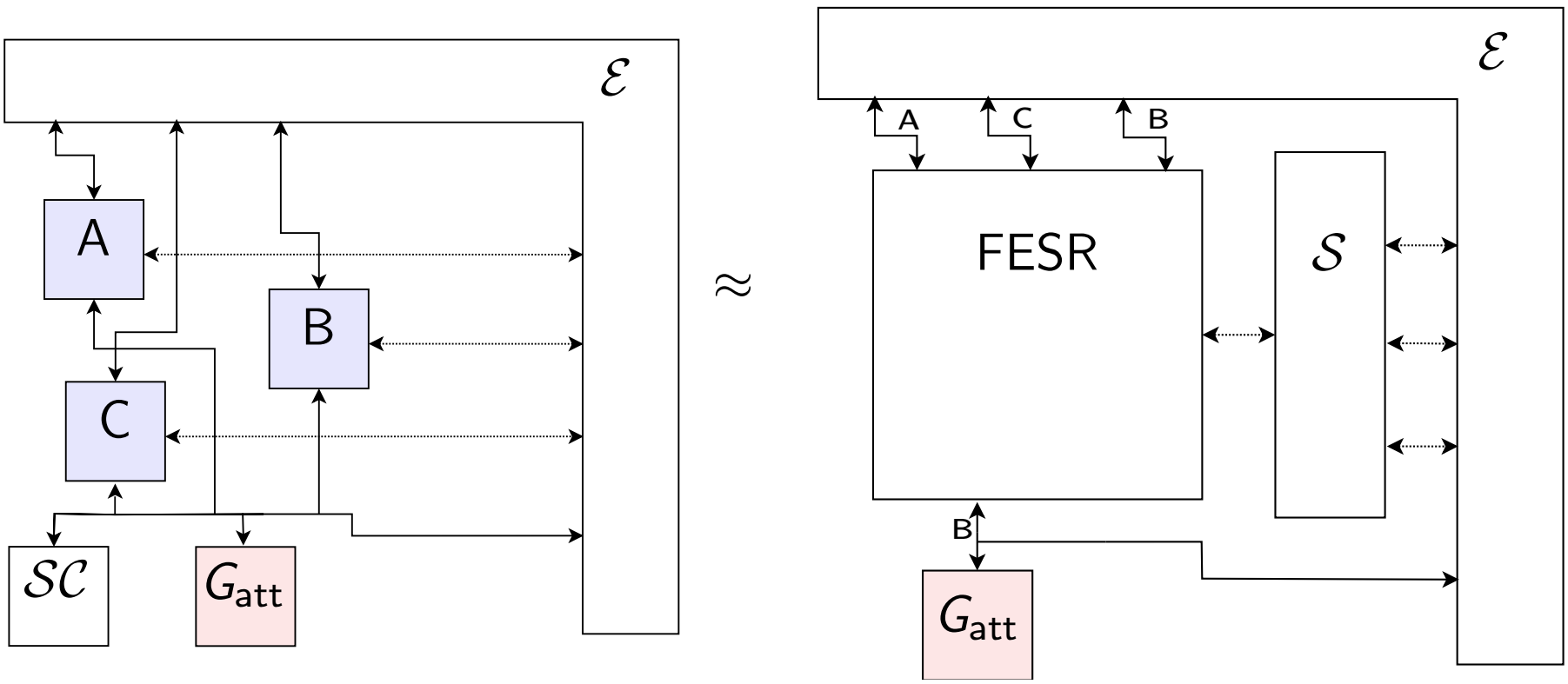
$(y, s') \xleftarrow{r} F(Dec(\mathsf{msk}, \mathsf{ct}), s)$

$s \leftarrow$ s'

Dec

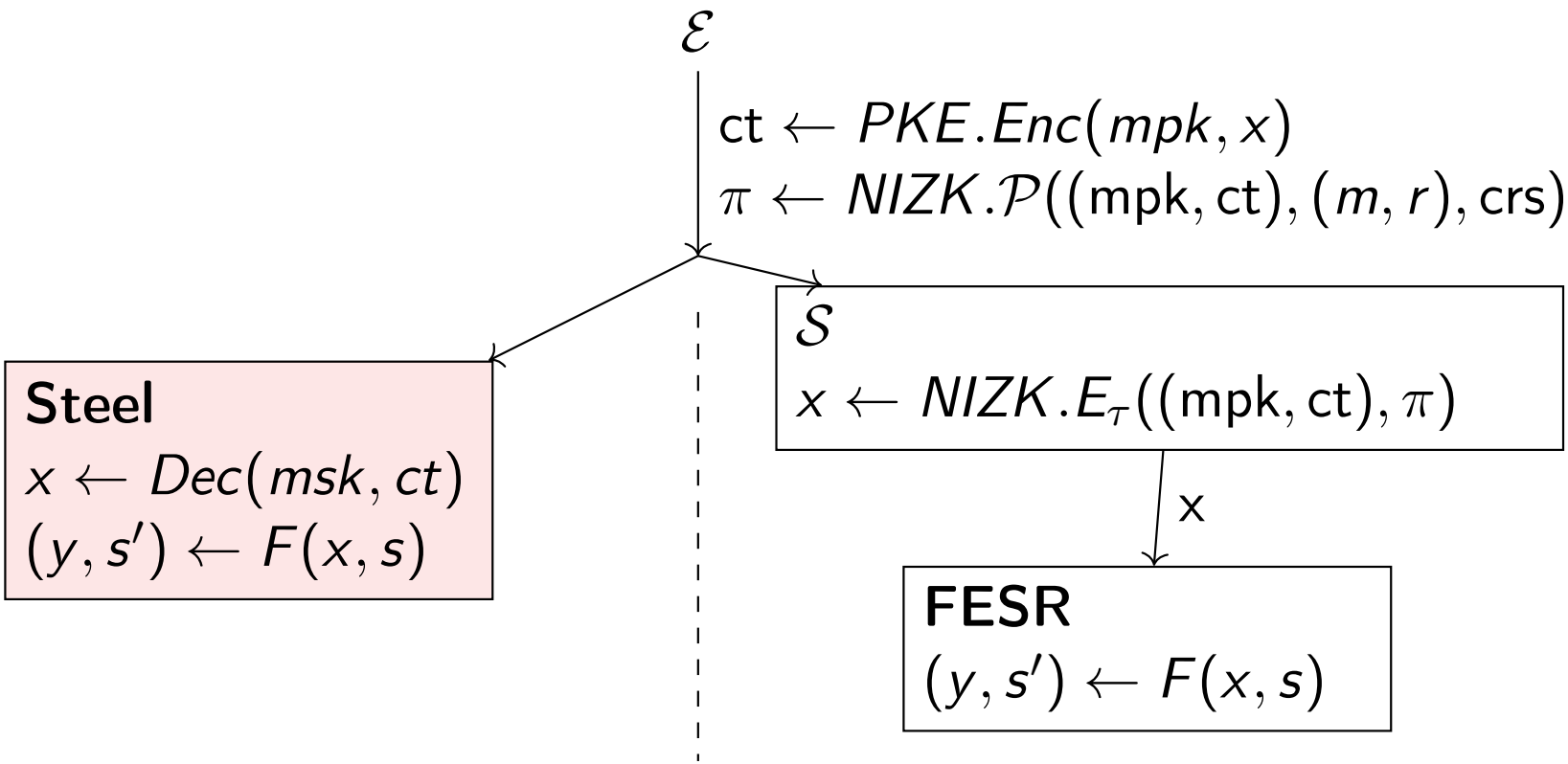
Proof

- Simulation in the UCGS setting (Badertscher et al, 2020)
- Identity bound on G_{att}
- Attestation Anonymity



How to simulate

- NIZK extraction
- Evaluation Backdoor

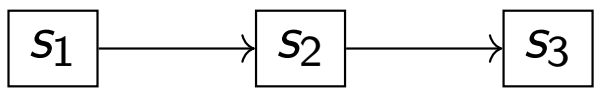


Assumptions

- EU-CMA for G_{att} signature scheme
- CCA for inter-enclave communication
- NIZK simulation-sound extractability
- CPA for client message encryption

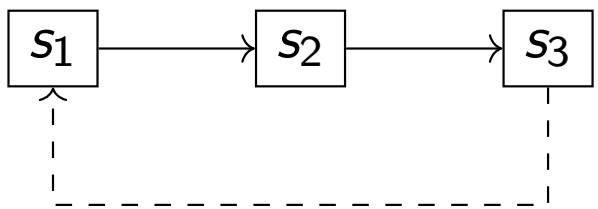
$G_{\text{att}}^{\text{rollback}}$ functionality

- Extend G_{att} to conduct rolling and forking attacks
- State is held in a tree
- On a resume call, the adversary can specify an arbitrary node



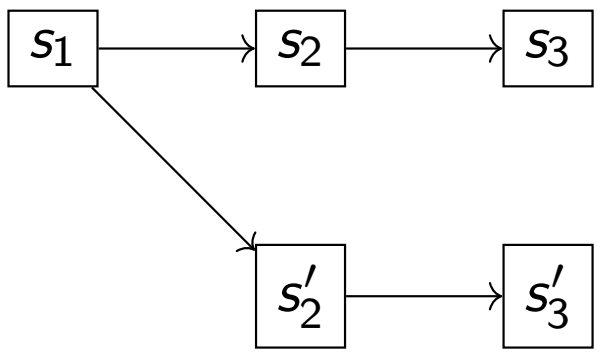
$G_{\text{att}}^{\text{rollback}}$ functionality

- Extend G_{att} to conduct rolling and forking attacks
- State is held in a tree
- On a resume call, the adversary can specify an arbitrary node



$G_{\text{att}}^{\text{rollback}}$ functionality

- Extend G_{att} to conduct rolling and forking attacks
- State is held in a tree
- On a resume call, the adversary can specify an arbitrary node



Mitigations

- Intel SGX Hardware Monotonic Counters
- Asynchronous counters
- Network protocols (ROTE, LCM)
- Building stateless enclaves
- Steel: rollback protect the Decryption Enclave

Conclusion

We strenghten FE to compute a larger class of functions efficiently

We model cryptographic protocols that use TEEs in a composable manner

We point out the limitations of TEEs once rollback and forking attacks are introduced

Thank you!

lorenzo.martinico@ed.ac.uk

https://ia.cr/2021/269