Formal Abstractions for Attested Execution Secure Processors

Eurocrypt
May 1st, 2017

Rafael Pass, Elaine Shi, Florian Tramèr
Trusted hardware:
Different communities, different world views
Trusted hardware:
Different communities, different world views

Crypto

Architecture + Systems & Security
Trusted hardware:
Different communities, different world views

- “Minimal” trusted hardware to circumvent theoretical impossibilities
- Little concern about practical performance
Trusted hardware:
Different communities, different world views

- “Minimal” trusted hardware to circumvent theoretical impossibilities
- Little concern about practical performance

- Trusted execution of "general-purpose" user-defined progs
- Cost-effectiveness, reusability, expressivity
Architecture community converged on "attested execution"

Bastion
Ascend
Aegis
XOM

GhostRider
Iso-X
Sanctum
Phantom

Academia

TPM
Intel® SGX
TrustZone®

Industry
Architecture community converged on “attested execution”

What is “attested execution”?  
What can it (not) express?
Attested Execution

Client → Compute prog on inp → Server
Attested Execution

Client

Compute \texttt{prog on inp}

Server

Enclave
Attested Execution

Client
- Verify

Server
- Enclave
  - Sign

Manufacturer

Compute `prog on inp`
Attested Execution

Client

Verify

Compute \texttt{prog} on \texttt{inp}

\texttt{outp}, \sigma

Server

Enclave

Sign

Attestation that \texttt{outp} is correctly computed from \texttt{prog} and \texttt{inp}

Manufacturer
Why Ideal Abstractions?
Why Ideal Abstractions?

- **Formal security proofs for implementations from precise abstractions and security models**
Why Ideal Abstractions?

• **Formal security proofs for implementations** from precise abstractions and security models

• **Ultimate Goal:** Formally verified processor implementing this formal abstraction
Formal Model

Signature scheme \( G_{\text{att}}[\Sigma, \text{reg}] \) Registry of all platforms with trusted hardware
Formal Model

$G_{\text{att}}[\Sigma, \text{reg}]$

Signature scheme
Registry of all platforms with trusted hardware

\begin{align*}
\text{init()}: & \quad \text{KeyGen}(1^\lambda) \\
\text{getpk}() & \text{ from } P: \text{ send to } P
\end{align*}
Informal Model

**$G_{\text{att}}[\Sigma, \text{reg}]$**

**init():**  
$\Sigma.\text{KeyGen}(1^\lambda)$

**getpk()** from $P$: send to $P$

**install**(prog, sid) from $P \in \text{reg}$:

---

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Formal Model

$G_{\text{att}}[\Sigma, \text{reg}]$

**Signature scheme**

**Registry of all platforms with trusted hardware**

**init()**: $\Sigma.$KeyGen($1^\lambda$)

**getpk()** from P: send $\rightarrow$ to P

**install**(prog, sid) from P $\in$ reg:

<table>
<thead>
<tr>
<th>enclave id (nonce)</th>
<th>enclave memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(eid, P)</td>
<td>(sid, prog, M)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Formal Model

\[ G_{\text{att}}[\Sigma, \text{reg}] \]

- **Signature scheme**
- **Registry of all platforms with trusted hardware**

**init()**: \( \Sigma.\text{KeyGen}(1^\lambda) \)

**getpk()** from \( P \): send \( \) to \( P \)

**install**(prog, sid) from \( P \in \text{reg} \):

**resume**(eid, inp) from \( P \in \text{reg} \):

<table>
<thead>
<tr>
<th>enclave id (nonce)</th>
<th>enclave memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>(eid, P)</td>
<td>(sid, prog, M)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
**Formal Model**

\[ G_{\text{att}}[\Sigma, \text{reg}] \]

**Initiation (\text{init()})**: \[ \Sigma.\text{KeyGen}(1^\lambda) \]

**Get Public Key (\text{getpk()})**: Get public key from P and send it to P.

**Install (\text{install()})**: Install program (\text{prog}) and session id (\text{sid}) from P in \text{reg}.

**Resume (\text{resume()})**: Resume session (\text{eid}, \text{inp}) and get output (\text{out}, M') via program execution (\text{prog}(\text{inp}, M)).
Formal Model

$G_{\text{att}}[\Sigma, \text{reg}]$

**Signature scheme**

**Registry of all platforms with trusted hardware**

\[ \text{init}(): \quad \Sigma.\text{KeyGen}(1^\lambda) \]

\[ \text{getpk}() \text{ from } P: \text{ send } \quad \text{to } P \]

\[ \text{install}(\text{prog}, \text{sid}) \text{ from } P \in \text{reg}: \]

\[ \text{resume}(\text{eid}, \text{inp}) \text{ from } P \in \text{reg}: \]

\[ (\text{out}, M') = \text{prog}(\text{inp}, M) \]
Formal Model

\[ g_{\text{att}}[\Sigma, \text{reg}] \]

**Signature scheme**

**Registry of all platforms with trusted hardware**

\[ \Sigma.\text{KeyGen}(1^\lambda) \]

**init()**: \( \Sigma.\text{KeyGen}(1^\lambda) \)

**getpk()** from P: send key to P

**install** \((\text{prog}, \text{sid})\) from P \(\in\) reg:

**resume** \((\text{eid}, \text{inp})\) from P \(\in\) reg:

\[ (\text{out}, M') = \text{prog}(\text{inp}, M) \]

\[ \sigma = \Sigma.\text{Sign}(\text{key}, \text{eid}, \text{sid}, \text{prog}, \text{out}) \]

send \((\text{out}, \sigma)\) to P
Composability with Global State
Composability with Global State

Model $\mathcal{G}_{\text{att}}$ as \textit{global} ideal functionality [CDPW’07]
Composability with Global State

Model $\mathcal{G}_{\text{att}}$ as \textit{global} ideal functionality [CDPW'07]

Attestation key is \textit{shared} across protocols
Composability with Global State

Model $\mathcal{G}_{\text{att}}$ as \textit{global} ideal functionality [CDPW’07]
Composability with Global State

Model $G_{\text{att}}$ as global ideal functionality [CDPW’07]

Example of concrete security issue: Non-deniability for parties in \text{reg}
The more interesting question

What is “attested execution”?

What can it (not) express?
The good

Powerful Abstraction!
The good

Powerful Abstraction!

\( G_{\text{att}} \rightarrow "\text{Stateful Obfuscation}" 

Impossible even with stateless tokens and cryptographic obfuscation
The good

Powerful Abstraction!

$G_{\text{att}} \rightarrow \"\text{"Stateful Obfuscation\"}\"
Impossible even with stateless tokens and cryptographic obfuscation

The surprise

UC-Secure MPC?
The good

Powerful Abstraction!

\( G_{\text{att}} \rightarrow \text{"Stateful Obfuscation"} \)
Impossible even with stateless tokens and cryptographic obfuscation

The surprise

UC-Secure MPC?

\[ \checkmark \text{ It’s Complicated } \]
Powerful Abstraction!

$G_{\text{att}} \rightarrow \text{“Stateful Obfuscation”}$

Impossible even with stateless tokens and cryptographic obfuscation

UC-Secure MPC?

☑ It’s Complicated
Consider 2PC
Consider 2PC

UC-secure 2PC possible if both parties have trusted hardware
Consider 2PC

UC-secure 2PC possible if both parties have trusted hardware

Impossible if only one party has trusted hardware!
Consider 2PC

This is counter-intuitive.

Impossible if only one party has trusted hardware!
Issue: non-deniability
Issue: non-deniability

Convinced that some honest party in the registry participated in the protocol

σ under global pk
Non-issue if all nodes have trusted hardware or if pk isn’t global

Convinced that some honest party in the registry participated in the protocol
What if we really really want to use a single trusted processor?
What if we really really want to use a single trusted processor?

**Extra setup assumption: Augmented CRS**
What if we really really want to use a single trusted processor?

Extra setup assumption: Augmented CRS

UC-Secure MPC with $O(1)$ crypto operations
What if we really want to use a single trusted processor?

Extra setup assumption: Augmented CRS

UC-Secure MPC with $O(1)$ crypto operations

Backdoor enclave program: allow simulator to extract inputs and program the outputs for corrupt parties
What if we really really want to use a single trusted processor?

\[\text{Server} \]

\[\text{prog}[f, g_{\text{acrs}}, \mathcal{P}_1 \ldots \mathcal{P}_n] \]

\[\mathcal{P}_i\]
What if we really really want to use a single trusted processor?

Server

\[ \text{prog}[f, G_{\text{acrs}}, P_1 \ldots P_n] \]

1. Generate \( pk_i, sk_i \)
What if we really really want to use a single trusted processor?

Server

\[ \text{prog}[f, G_{\text{acrs}}, P_1 \ldots P_n] \]

1. Generate \( pk_i, sk_i \)

\[ \text{pk}_i, \sigma \]

Full protocol replaces \( \sigma \) by a WI-Proof

\[ P_i \]
What if we really really want to use a single trusted processor?

Server

\[ \text{prog}[f, G_{\text{acs}}, P_1 \ldots P_n] \]

1. Generate \( pk_i, sk_i \)

Full protocol replaces \( \sigma \) by a WI-Proof

\( pk_i, \sigma \)

Key-exchange

\( P_i \)
What if we **really really** want to use a single trusted processor?

Server

\[ \text{prog}[f, G_{\text{acs}}, P_1 \ldots P_n] \]

1. Generate \( pk_i, sk_i \)

1. Collect all \( inp_i \)

**Full protocol replaces \( \sigma \) by a WI-Proof**

\( pk_i, \sigma \)

Key-exchange

Encrypted \( inp_i \)

\( P_i \)
What if we really really want to use a single trusted processor?

Server

\[ \text{prog}[f, G_{\text{acrs}}, P_1 \ldots P_n] \]

1. Generate \( pk_i, sk_i \)

1. Collect all \( inp_i \)

2. Compute \( outp^* \)

Full protocol replaces \( \sigma \) by a WI-Proof

\( pk_i, \sigma \)

Key-exchange

Encrypted \( inp_i \)

Encrypted \( outp_i \)
What if we really really want to use a single trusted processor?

3. Trapdoors

$$\text{prog} \left[f, G_{\text{acrs}}, P_1 \ldots P_n \right]$$
What if we really really want to use a single trusted processor?

- Server

  \[\text{prog}[f, G_{\text{acrs}}, P_1 \ldots P_n]\]

  3. Trapdoors

  \[\text{check}(G_{\text{acrs}}, P_i, \text{id}_i)\]

- Sim

  \[\text{extract}(\text{id}_i)\]
What if we really really want to use a single trusted processor?

3. Trapdoors

\[
\text{check}(G_{acrs}, P_i, id_i)
\]

\[
\text{extract}(id_i)
\]

\[
\text{sk}_i
\]

\[
\text{Sim can recover } inp_i
\]
What if we really really want to use a single trusted processor?

3. Trapdoors

\[
\text{prog}[f, g_{\text{acrs}}, \mathcal{P}_1 \ldots \mathcal{P}_n]
\]

\[
\text{extract}(id_i)
\]

\[
\text{equivocate}(id_i, v)
\]

\[
\text{check}(g_{\text{acrs}}, \mathcal{P}_i, id_i)
\]

set \text{outp}_i = v
Fair 2PC
• Fairness impossible for general functionalities in plain model [Cleve86]

Fair 2PC
Can trusted hardware help with fairness?

• Fairness impossible for general functionalities in plain model [Cleve86]

Fair 2PC
UC-Secure Fair 2PC

Enhanced model: Clock-aware secure processor
UC-Secure Fair 2PC

Enhanced model: Clock-aware secure processor

• Fair 2PC possible if both parties have clock-aware secure processors
UC-Secure Fair 2PC

Enhanced model: Clock-aware secure processor

- Fair 2PC possible if both parties have clock-aware secure processors
- Fair coin-tossing possible if one party has clock-aware secure processors (+ ACRS)
UC-Secure Fair 2PC

Enhanced model: Clock-aware secure processor

• Fair 2PC possible if both parties have clock-aware secure processors

• Fair coin-tossing possible if one party has clock-aware secure processors (+ ACRS)
Enclaves establish secure channel
Enclaves establish secure channel

Enclaves exchange inputs and compute outputs
Enclaves establish secure channel

Enclaves exchange inputs and compute outputs

“Will release to Alice in $2^\lambda$ time”

“Will release to Bob in $2^\lambda$ time”
Enclaves establish secure channel

Enclaves exchange inputs and compute outputs

“Will release to Alice in $2^\lambda$ time”

“Will release to Bob in $2^\lambda$ time”

“Will release to Alice in $2^{\lambda-1}$ time”

“Will release to Bob in $2^{\lambda-1}$ time”

...
Enclaves establish secure channel

If Alice learns result at time $t < 2^\lambda$, Bob will learn it at the latest by time $2t$

+ no “wasted” computation!

“Will release to Alice in $2^{\lambda-1}$ time”

“Will release to Bob in $2^{\lambda-1}$ time”
What next?

Formal abstractions of trusted hw
What next?

Attested execution is a powerful assumption

\[\rightarrow\] Stateful Obfuscation, Efficient MPC, Fair 2PC

Formal abstractions of trusted hw
What next?

Attested execution is a powerful assumption

⟹ Stateful Obfuscation, Efficient MPC, Fair 2PC

Subtle issues unless all parties have trusted hardware

⟹ Non-deniability, Extra setup assumptions

Formal abstractions of trusted hw
What next?

Attested execution is a powerful assumption
⟹ Stateful Obfuscation, Efficient MPC, Fair 2PC

Subtle issues unless all parties have trusted hardware
⟹ Non-deniability, Extra setup assumptions

Formal abstractions of trusted hw
Formally verified secure processor design
What next?

Attested execution is a powerful assumption

⇒ Stateful Obfuscation, Efficient MPC, Fair 2PC

Subtle issues unless *all parties have trusted hardware*

⇒ Non-deniability, Extra setup assumptions

- Formal abstractions of trusted hw
- Formally verified secure processor design
- Secure implementations from formally secure abstractions
Formal abstractions of trusted hw
Formally verified secure processor design
Secure implementations from formally secure abstractions

Thank You