Trust Engineering via Crypto Protocols

Joshua Guttman
Ian Kretz Andrew Lilley-Brinker John Ramsdell
The MITRE Corporation

Hot Topics in the Science of Security
The University of Kansas
23 Sept. 2020

MITRE

Trust Engineering

- System participants have varying goals
  - My goals constrain my interactions
  - Choices require information about peers
- Crypto protocols propagate trust data
  - Authentication, authorization decisions, exclusive access, attested code in enclave
- Trust engineering means designing systems so that:

Each decision based on
definite assumptions and reliable conclusions
about peers

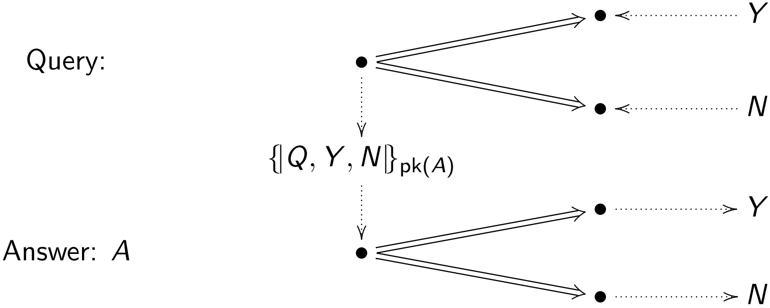
A simplest example

The yes-or-no protocol

- Goal: Ask a yes-or-no question, get answer from peer
  - Question and answer cryptographically protected
  - Even an adversary who guesses the question doesn't learn answer
- Only use one encryption

Yes-or-no protocol

Choose random values Y, N, encrypt together with question Q



A simplest example

The yes-or-no protocol

- Goal: Ask a yes-or-no question, get answer from peer
  - Question and answer cryptographically protected
  - Even an adversary who guesses the question doesn't learn answer

Could have chosen random values Y, N
in other order

- Only use one encryption

A simplest example

The yes-or-no protocol

- Goal: Ask a yes-or-no question, get answer from peer
  - Question and answer cryptographically protected
  - Even an adversary who guesses the question doesn't learn answer

Could have chosen random values Y, N
in other order

- Only use one encryption
- Random Y, N don't say yes or no

Structure of protocol propagates answer

MITRE

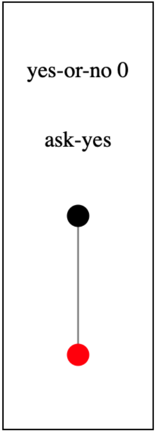
Analyzing Yes-or-No

- If query $\{|Q, Y, N|\}_{\mathsf{pk}(A)}$ receives answer Y,
  what else must have happened in distributed system?

Analyzing Yes-or-No

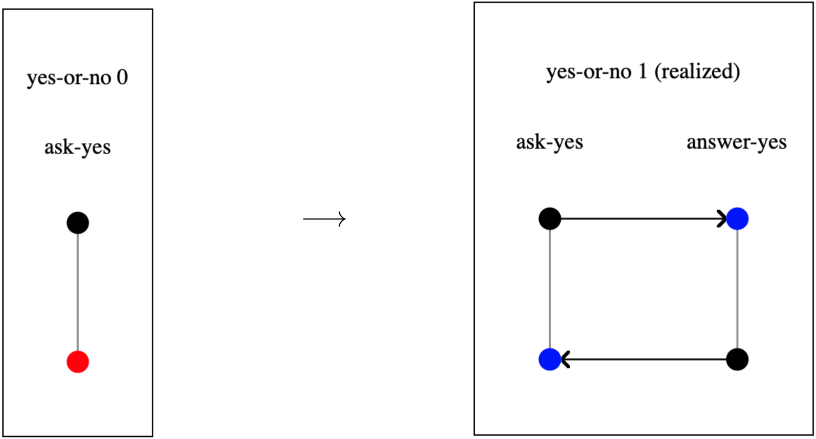
- If query $\{\!|Q, Y, N|\!\}_{\mathsf{pk}(A)}$ receives answer Y,
  what else must have happened in distributed system?
- Assume decryption key $\mathsf{pk}(A)^{-1}$ uncompromised
- Conclude answerer A sent Y

Analysis of Yes-or-No via CPSA: Hearing Y



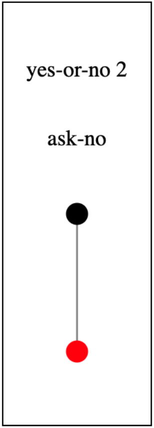
$\mathsf{pk}(A)^{-1}$ non-compromised $\quad\quad Y$ fresh

Analysis of Yes-or-No via CPSA: Hearing Y



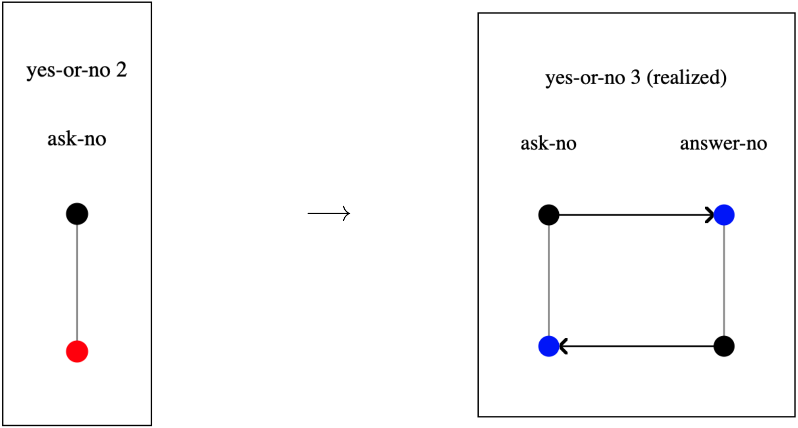
$\mathrm{pk}(A)^{-1}$ non-compromised $\quad Y$ fresh

Analysis of Yes-or-No via CPSA: Hearing N



$\mathsf{pk}(A)^{-1}$ non-compromised $\qquad$ N fresh

Analysis of Yes-or-No via CPSA: Hearing N



$\mathrm{pk}(A)^{-1}$ non-compromised $\quad$ N fresh

Analyzing Yes-or-No

- If query $\{\!|Q, Y, N|\!\}_{\mathsf{pk}(A)}$ receives answer Y, what else must have happened in distributed system?
- Assume decryption key $\mathsf{pk}(A)^{-1}$ uncompromised
- Conclude answerer A sent Y
- Cryptographic Protocol Shapes Analyzer solves:

If some scenario has occurred, what
minimal, essentially different executions are possible?

Protocol analysis vs. trust

- Protocol analysis tells us:
  - What must have happened elsewhere — authentication
  - What cannot have happened elsewhere — confidentiality
  - What assumptions underlie conclusions
- Trust provides reasons for assumptions, e.g.:
  - Organizational practices
  - Interests of real-world principals
  - Safety from authorization policies
- Each may amplify the other
  - Trust in known CA helps authenticate server
  - Protocol conclusions attribute claims to principals

MITRE

Trust Engineering

- System participants have varying goals
  - My goals constrain my interactions
  - Choices require information about peers
- Crypto protocols propagate trust data
  - Authentication, authorization decisions, exclusive access, attested code in enclave
- Trust engineering means designing systems so that:

Each decision based on
definite assumptions and reliable conclusions
about peers

MITRE

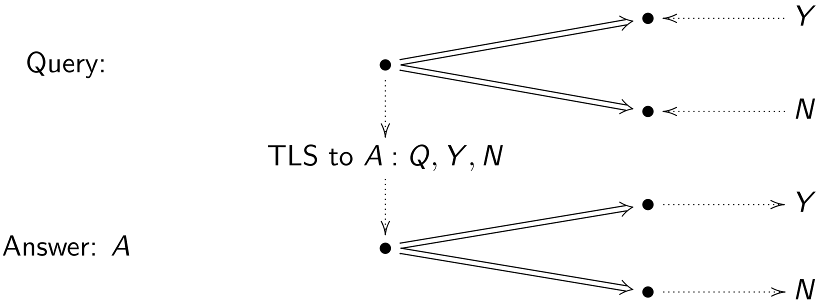
Isn't protocol design dangerous?

- Objection: "Protocol design is not for me. I stick with TLS"
- Reply: Sure, use TLS for:
  - Secure channels
  - In-flight encryption
- Still need:
  - Digital signature (non-repudiability)
  - Decisions what to send
  - Design for key distribution and transactions

MITRE

Yes-or-no protocol over TLS

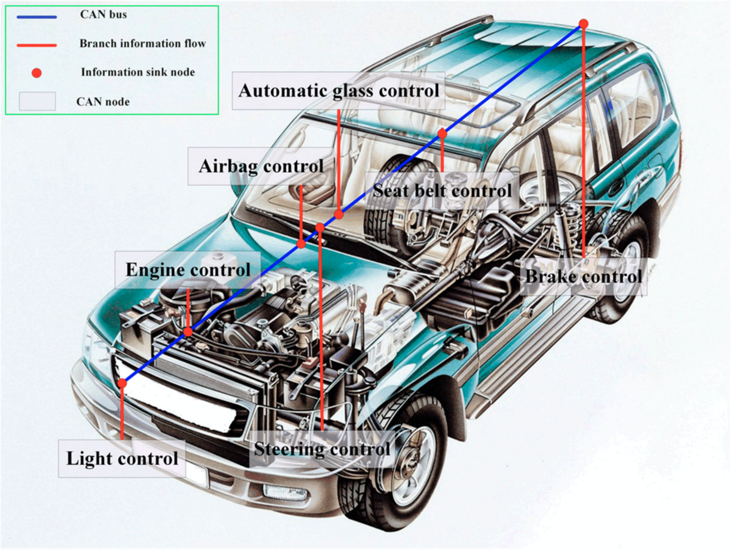
Choose random values Y, N, encrypt together with question Q



Isn't protocol design dangerous?

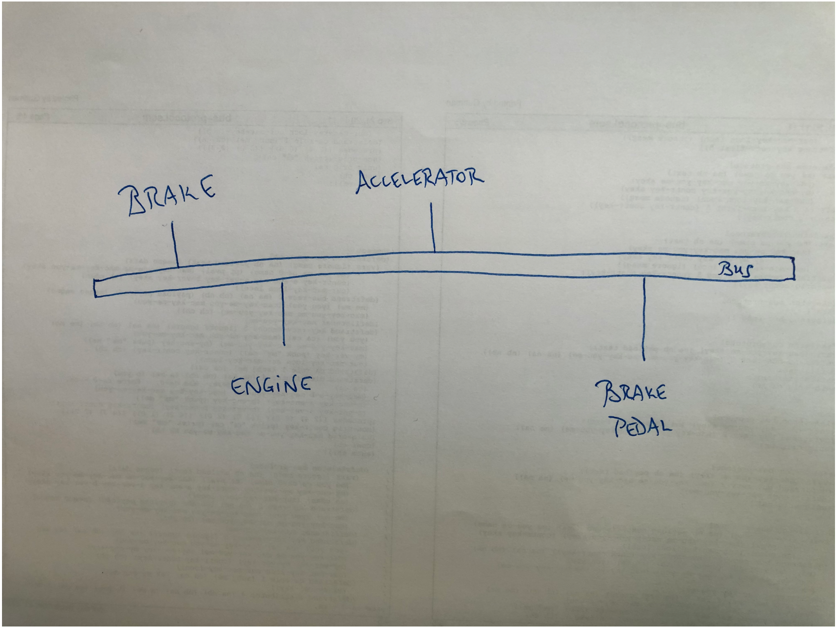
- Objection: "Protocol design is not for me. I stick with TLS"
- Reply: Sure, use TLS for:
  - Secure channels
  - In-flight encryption
- Still need:
  - Digital signature (non-repudiability)
  - Decisions what to send
  - Design for key distribution and transactions
- CPSA offers a secure channel abstraction for this

More interesting example: A data bus



https://www.researchgate.net/publication/334969096_Anomaly_Detection_of_CAN_Bus_Messages_Using_A_Deep_Neural_Network_for_Autonomous_Vehicles/figures?lo=1,

Schematically...

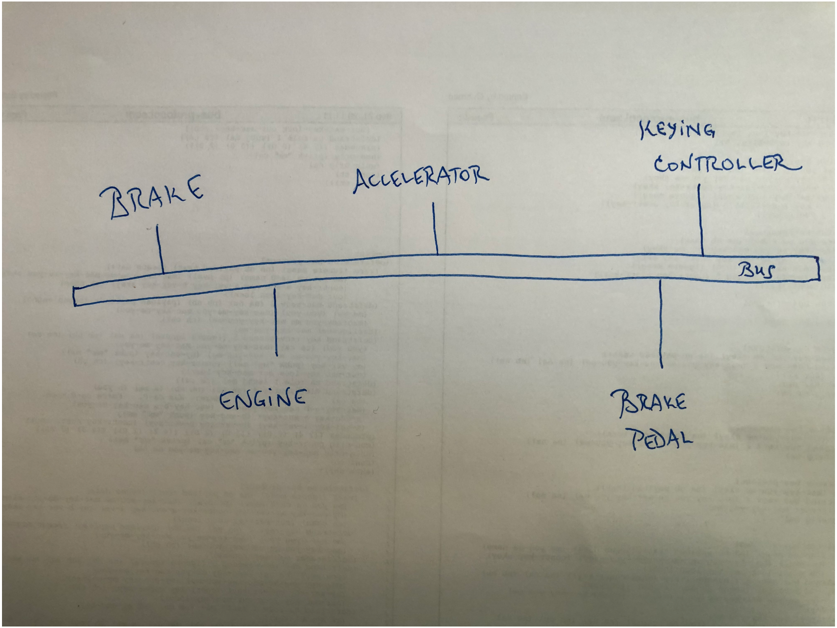


MITRE

Security considerations

- Most msgs don't need confidentiality
- Entertainment system should never
  - send control msgs to brakes
  - generate msgs purportedly from brake pedal
  - share authentication secret for pedal-to-brake msgs
- Hence: distribute pairwise Message Authentication Codes
  - Centralize authorization policy
  - Distribute shared secrets only to authorized pairs

With keying for Message Authentication Codes



MITRE

Designing the protocol

- Device behaviors:
  - Receive MAC keys for a peer device
  - Send or receive MACed msgs
- Controller behavior:
  - Deliver MAC keys to permitted peers
- Protocol considerations:
  - Long-term protection to deliver MAC keys
  - Certs for long-term keys
  - Devices store MAC keys in state, retrieve them for use

MAC key distribution: Controller r

$$
\begin{array}{lll}
\bullet \leftarrow\cdots & [\![\, \mathtt{cert}\ \ A \mathbin{\hat{}} \mathsf{pk}(A) \,]\!]_{\mathsf{sk}_{\mathtt{ca}}(CA)} & = c_1 \\
\Downarrow & & \\
\bullet \leftarrow\cdots & [\![\, \mathtt{cert}\ \ B \mathbin{\hat{}} \mathsf{pk}(B) \,]\!]_{\mathsf{sk}_{\mathtt{ca}}(CA)} & = c_2 \\
\Downarrow & & \\
\bullet \cdots\rightarrow & \{\!|\mathtt{mkp}\ \ [\![\, \mathtt{del}\ \ A \mathbin{\hat{}} B \mathbin{\hat{}} r \mathbin{\hat{}} mk_1 \mathbin{\hat{}} mk_2 \mathbin{\hat{}} c_1 \mathbin{\hat{}} c_2 \,]\!]_{\mathsf{sk}_{\mathtt{sig}}(r)} |\!\}_{\mathsf{pk}(A)} & \\
\Downarrow & & \\
\bullet \cdots\rightarrow & \{\!|\mathtt{mkp}\ \ [\![\, \mathtt{del}\ \ B \mathbin{\hat{}} A \mathbin{\hat{}} r \mathbin{\hat{}} mk_2 \mathbin{\hat{}} mk_1 \mathbin{\hat{}} c_2 \mathbin{\hat{}} c_1 \,]\!]_{\mathsf{sk}_{\mathtt{sig}}(r)} |\!\}_{\mathsf{pk}(B)} &
\end{array}
$$

MAC key distribution: Device A

$$\bullet \leftarrow \{\!|\texttt{mkp}\ [\![\texttt{del}\ \ A \,\hat{}\, B \,\hat{}\, r \,\hat{}\, mk_1 \,\hat{}\, mk_2 \,\hat{}\, c_1 \,\hat{}\, c_2\,]\!]_{\mathsf{sk}_{\texttt{sig}}(r)} |\!\}_{\mathsf{pk}(A)}$$

$$\Downarrow$$

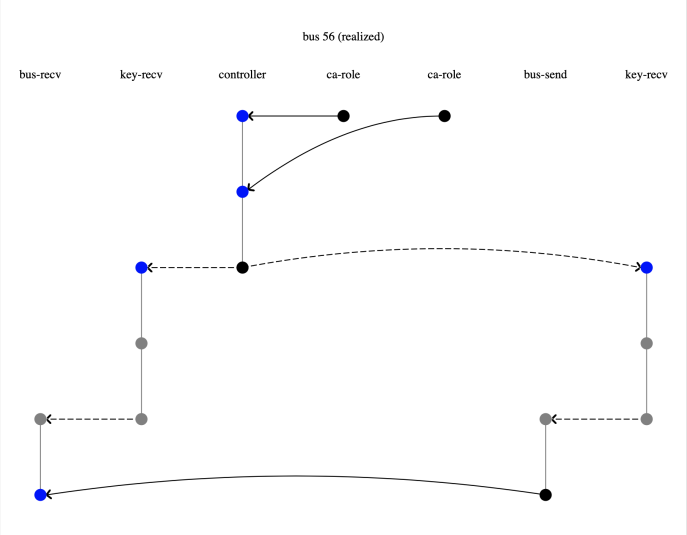
$$\bullet \leftarrow x \qquad \text{trans}$$

$$\rightarrow \texttt{mkr}\ \ A \,\hat{}\, B \,\hat{}\, r \,\hat{}\, mk_1 \,\hat{}\, mk_2 \,\hat{}\, c_1 \,\hat{}\, c_2$$

Message reception: Device A

$$\begin{tikzcd}
\circ \arrow[d, Rightarrow] & \arrow[l, dotted, "\mathrm{obsv}"'] \texttt{mkr}\ \ A \mathbin{\hat{}} B \mathbin{\hat{}} r \mathbin{\hat{}} mk_1 \mathbin{\hat{}} mk_2 \mathbin{\hat{}} c_1 \mathbin{\hat{}} c_2 \\
\bullet & \arrow[l, dotted] m \mathbin{\hat{}} B \mathbin{\hat{}} i \mathbin{\hat{}} \#(\texttt{mh}\ \ m \mathbin{\hat{}} B \mathbin{\hat{}} i \mathbin{\hat{}} mk_2)
\end{tikzcd}$$

A CPSA result: A reception



$\mathsf{dk}(A)$, $\mathsf{dk}(B)$, $\mathsf{sk}_{\mathtt{ca}}(CA)$, $\mathsf{sk}_{\mathtt{sig}}(r)$ all uncompromised

MITRE

How could we have done it wrong?

Well, we did it wrong repeatedly

- Bad MAC key packaging
- Key direction mismatch
- Didn't deliver certs with MAC keys to devices
  - What CA certified peer's long term key?
  - Untrustworthy CA could certify compromised long-term key
    - Controller uses compromised long-term key
    - MAC keys disclosed when distributed
  - CPSA results motivated improvement
  - Trust distinction: Trust one CA vs. trust all CAs

Making trust reasoning explicit

- Authorization policy
- Reasons for thinking keys:
  - Undisclosed
  - Used only as dictated by this protocol
- Reasons for thinking principals:
  - Can protect keys
  - Adhere to protocol

Principals = People or hardware or software

Making authorization policy explicit

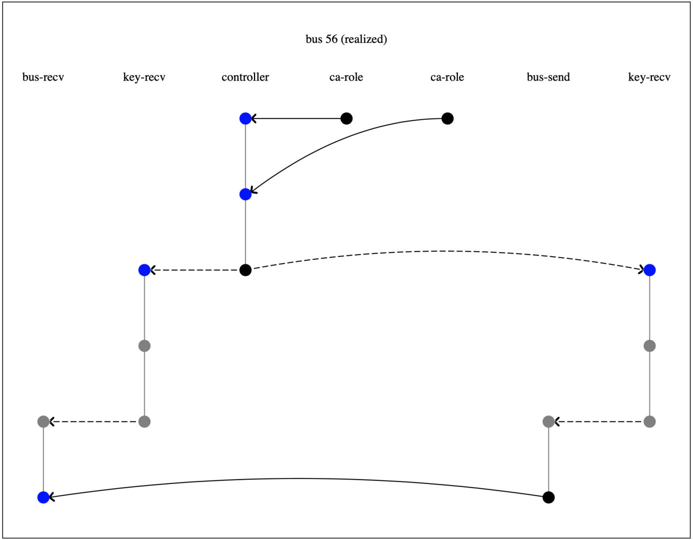
- Enrich protocol reasoning with rules, eg:

```
(defrule controller-respects-authorization
  (forall
   ((z1 strd) (ctr a b name))
   (implies
    (and (p "controller" z1 3)
         (p "controller" "ctr" z1 ctr)
         (p "controller" "a" z1 a)
         (p "controller" "b" z1 b))
    (fact policy-permits ctr a b))))
```

Can also reflect RBAC, XACML etc.

MITRE

Updated result




```
(facts (policy-permits ctr you me) ...)
```

MITRE

Making non-compromise explicit

- Simple approach:
  - CA ensures known individual possesses key
  - Self-protection ensures individual protects it
  - Threat intelligence determines if key stolen

MITRE

Persistent safety

```
(defrule persistent-safety
  (forall
   ((k mesg) (subj name))
   (implies
    (and (fact starts-safe subj k)
         (fact keeps-safe subj k))
    (non k))))

(defrule ca-trust-anchor
  (forall
   ((z1 strd) (subj ca name))
   (implies
    (and (p "ca-role" z1 1)
         (p "ca-role" "subj" z1 subj)
         (p "ca-role" "ca" z1 ca)
         (non (privk "ca" ca)))
    (fact starts-safe subj (privk "enc" subj)))))
```

MITRE

Threat-aware controller

```
(defrule controller-threat-aware-1
  (forall
   ((z1 strd) (ctr a name))
   (implies
    (and (p "controller" z1 3)
         (p "controller" "ctr" z1 ctr)
         (p "controller" "a" z1 a)
         (fact threat-aware ctr))
    (fact keeps-safe a (privk "enc" a)))))
```

Reasoning about attestation

Building atop SGX

- SGX: security services for enclaves within user processes

  confidentiality: code, data encrypted whenever evicted

  attestation: other entities can ascertain enclave's

  - code
  - selected data — esp. public key

Reasoning about attestation

Building atop SGX

- SGX: security services for enclaves within user processes

  confidentiality: code, data encrypted whenever evicted

  attestation: other entities can ascertain enclave's

  - code
  - selected data esp. public key

- This can be a big deal:

  Protect enclave secrets, allowing

  Secure channels between components running

  Known code, all

  Independent of vulnerable lower levels

  e.g. operating system unexpected hardware sysadmins

MITRE

Reasoning about attestation

Building atop SGX

- SGX: security services for enclaves within user processes

  confidentiality: code, data encrypted whenever evicted

  attestation: other entities can ascertain enclave's

  - code
  - selected data esp. public key

- This can be a big deal:

  Protect enclave secrets, allowing

  Secure channels between components running

  Known code, all

  Independent of vulnerable lower levels

  e.g. operating system unexpected hardware sysadmins

although with limitations. . .

MITRE

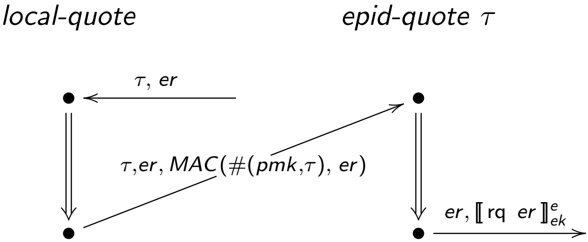
SGX: How it provides attestation

- Enclave Record includes:
  - Enclave id
  - Hash of controlling code
  - Message, in our usage always including public key
  - Many supplementary fields
- Processor provides local enclave attestation — MAC
- Quoting Enclave converts local quote to remote quote — EPID
- Intel: validates remote quotes online — Intel Attest. Serv.
  - ensures supply-chain origin
  - created runtime dependency on Intel

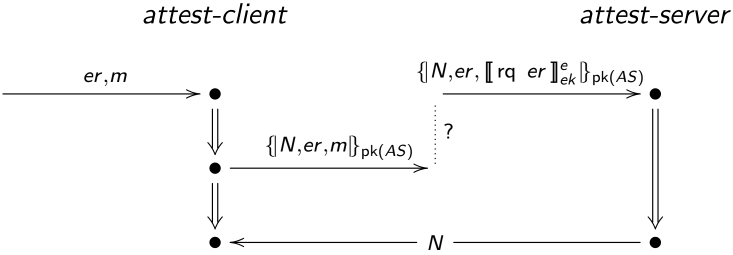
SGX: How it provides attestation

- Enclave Record includes:
  - Enclave id
  - Hash of controlling code
  - Message, in our usage always including public key
  - Many supplementary fields
- Processor provides local enclave attestation — MAC
- Quoting Enclave converts local quote to remote quote — EPID
- Intel: validates remote quotes online — Intel Attest. Serv.
  - ensures supply-chain origin
  - created runtime dependency on Intel
  - new alternative: attestation rooted in with non-Intel CA — ECDSA

SGX core roles, 1



SGX core roles, 2



Three types of rules

Hardware rules: processor requirements

- Local quote issued implies corresponding enclave
- Processor can protect core secrets

Trust rules: key generation and certification practices

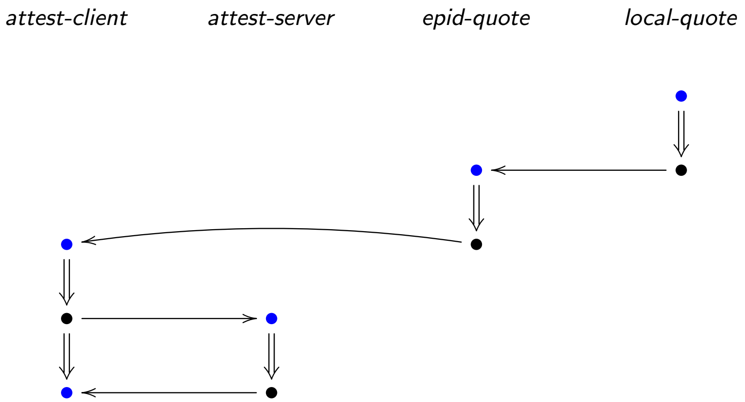
- Intel Attestation Server private key protected, compliant
- Accepted QE key generated in provisioning protocol

Attestation rules: behavioral requirements on application code

- User enclave makes fresh key pair
  - registers public key;   protects private key
  - uses private key only in accordance w/ application protocol

SGX desired execution

If *attest-client* runs with non-compromised AS



Facts: $\texttt{Enclave}(er, pmk)$ $\texttt{ManuMadeEpid}(ek)$

Non keys: $\texttt{Non}(\mathrm{dk}(AS))$ $\texttt{Non}(pmk)$ $\texttt{Non}(ek)$

SGX desired execution

If *attest-client* runs with non-compromised AS

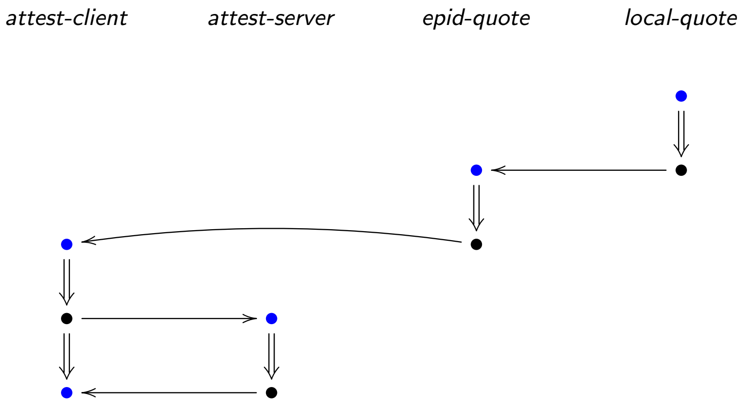
attest-client *attest-server* *epid-quote* *local-quote*

Facts:

Non keys: $\mathtt{Non}(\mathrm{dk}(AS))$

SGX desired execution

If *attest-client* runs with non-compromised AS



Facts: $\texttt{Enclave}(\mathit{er}, \mathit{pmk})$ $\texttt{ManuMadeEpid}(\mathit{ek})$

Non keys: $\texttt{Non}(\mathrm{dk}(AS))$ $\texttt{Non}(\mathit{pmk})$ $\texttt{Non}(\mathit{ek})$

Three types of rules

Hardware rules: processor requirements

- Local quote issued implies corresponding enclave
- Processor can protect core secrets

Trust rules: key generation and certification practices

- Intel Attestation Server private key protected, compliant
- Accepted QE key generated in provisioning protocol

Attestation rules: behavioral requirements on application code

- User enclave makes fresh key pair
  - registers public key;   protects private key
  - uses private key only in accordance w/ application protocol

Rule governing local quote

Quote guarantees enclave

Rule

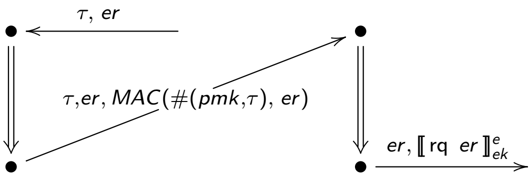
$$\forall z : \textrm{STRD}, \quad eid, ch, rest : \textrm{MESG}, \quad k : \textrm{AKEY}, \quad pmk : \textrm{SKEY}\,.$$

$$\begin{aligned}
&\texttt{LocQt}(z, 2) \quad \wedge \\
&\texttt{LocQtER}(z, eid :: ch :: k :: rest) \quad \wedge \\
&\texttt{LocQtPr}(z, pmk) \quad \wedge \quad \texttt{Non}(pmk) \\
&\Longrightarrow \\
&\qquad \texttt{EnclCodeKey}(eid, ch, k, pmk)
\end{aligned}$$

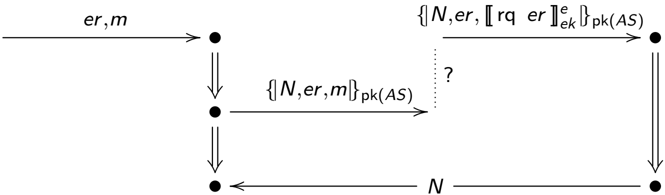
MITRE

SGX core roles

local-quote *epid-quote* τ



attest-client *attest-server*



Rule governing attest server

AS says EPID key is manufacturer-made and non-compromised

Rule

$$\begin{aligned}
&\forall z\colon \textsc{strd},\ ek : \textsc{akey}\,.\\
&\qquad \mathtt{AttServ}(z, 2) \quad \wedge\\
&\qquad \mathtt{ASQtKey}(z, ek)\\
&\qquad \Longrightarrow\\
&\qquad\qquad \mathtt{ManuMadeEpid}(ek) \quad \wedge \quad \mathtt{Non}(ek)
\end{aligned}$$

MITRE

Attestation rule for application level code

Rule

$$\begin{array}{l}\forall e, ch\colon \textsc{mesg},\ k\colon \textsc{akey},\ pmk\colon \textsc{skey}\,.\\ \quad \texttt{PeerCode}(ch) \quad \wedge \\ \quad \texttt{EnclCodeKey}(e, ch, k, pmk) \\ \Longrightarrow \\ \quad \texttt{Non}(k^{-1})\end{array}$$

Induces a behavioral requirement

`PeerCode`(ch) means code that hashes to ch:

- Must:
  - Freshly generate a keypair k, k^{-1}
  - Move k into enclave record
  - Use k^{-1} only in accordance with the protocol
- Must not disclose:
  - k^{-1}
  - Any computed value providing advantage on k^{-1}

Satisfying the behavioral requirement:
Why not compile code directly
from the CPSA spec?

MITRE

Trust Engineering

- System participants have varying goals
  - My goals constrain my interactions
  - Choices require information about peers
- Crypto protocols propagate trust data
  - Authentication, authorization decisions, exclusive access, attested code in enclave
- Trust engineering means
  designing systems so that:

  Each decision based on
  definite assumptions and reliable conclusions
  about peers

MITRE

Joshua Guttman
Ian Kretz Andrew Lilley-Brinker John Ramsdell
The MITRE Corporation

{guttman, ikretz, abrinker, ramsdell}@mitre.org