CSE 5095 & ECE 4451 & ECE 5451 – Spring 2017

Lecture 4b

- Slide deck extracted from Kamran's tutorial on SGX, presented during ECE 6095 Spring 2017 on Secure Computation and Storage, a precursor to this course

Crypto Background & Concepts SGX Software Attestation

Marten van Dijk

Syed Kamran Haider, Chenglu Jin, Phuong Ha Nguyen

Department of Electrical & Computer Engineering
University of Connecticut

With the help of:

1. Intel SGX Tutorial (Reference Number: 332680-002) presented at ISCA 2015
2. "*Intel SGX Explained*", Victor Costan and Srinivas Devadas, CSAIL MIT
3. Computer Security taught by Aggelos Kiayias, 2006-2014

Crypto Background & Concepts

- Public Key Cryptography:
  - Key Agreement
  - Encryption
  - Signatures
  - Certificate Authority
  - Key Management and Public Key Infrastructure

- Hashes and MACs

Key Agreement

- Secret-Key Crypto:
- Alice and Bob first agree on a secret key K
- Alice and Bob use a symmetric key encryption mechanism (such as AES) to encrypt their messages: AES_K(message)
- Here we have the computational security assumption that the inverse (decryption) of AES_K cannot be computed by an adversary, even with access to lots of resources
  - many computation cores,
  - lots of time,
  - some messages with their encryptions,
  - may be even adaptively chosen messages with their encryptions, …
- Perfect security, i.e. one only assumes an information theoretic (or statistical) security assumption, is only possible if $|K|=|message|$ and each key K can only be used for at most one encryption

Key Agreement

- For symmetric key crypto, a joint/shared secret key needs to be established first
  - Symmetric key crypto is very efficient
  - So, once a key is agreed upon (like an attestation key), then we want to use symmetric crypto !
  - Key agreement (cannot be public !!) may be based on (a physical trusted channel or) public key cryptograph concepts,
    - which is much less efficient and
    - requires an “agreed upon” or “certified” public key
    - needs trust in a public key infrastructure (trusted certificate authority) with trusted key management (as private/secret keys do leak since they are maintained by SW/HW which does have vulnerabilities)

A Group Theoretic Problem

- Given a finite multiplicative group
- Consider a "generator" g in this group generating cyclic subgroup $< g > = \{1, g, g^2, g^3, ..\}$
- Problem: Given h in $< g >$, given g, compute a solution x such that $h = g^x$.
- Difficult: Current state-of-the-art algorithms take subexponential time in the log of the order of the subgroup.
- It is easy to compute h from g and x !!

Diffie Hellman Key Agreement

- Alice chooses x_A, computes g^{x_A}, and transmits g^{x_A} to Bob
- Bob chooses x_B, computes g^{x_B}, and transmits g^{x_B} to Alice
- Now Alice and Bob are each able to compute (without any further help) $K = g^{x_A x_B}$

- Catch: How does Alice knows she has set up a key with Bob and not with a man-in-the-middle Eve?
- Alice and Bob need a pre-established authentic channel in the first place!

Public Key Crypto

- Alice wants to send a message to Bob
- Bob publishes his public key
- Alice obtains Bob's public key
  - in a trusted way !! Again authenticity must be verified.
  - The public key is signed by a trusted third party (a certificate authority)
- Alice encrypts the message with the public key → ciphertext
- The ciphertext is transmitted to Bob
- Bob decrypts the ciphertext with his secret key
  - The public key and secret key have an algebraic relationship which allows public key encryption together with secret key decryption

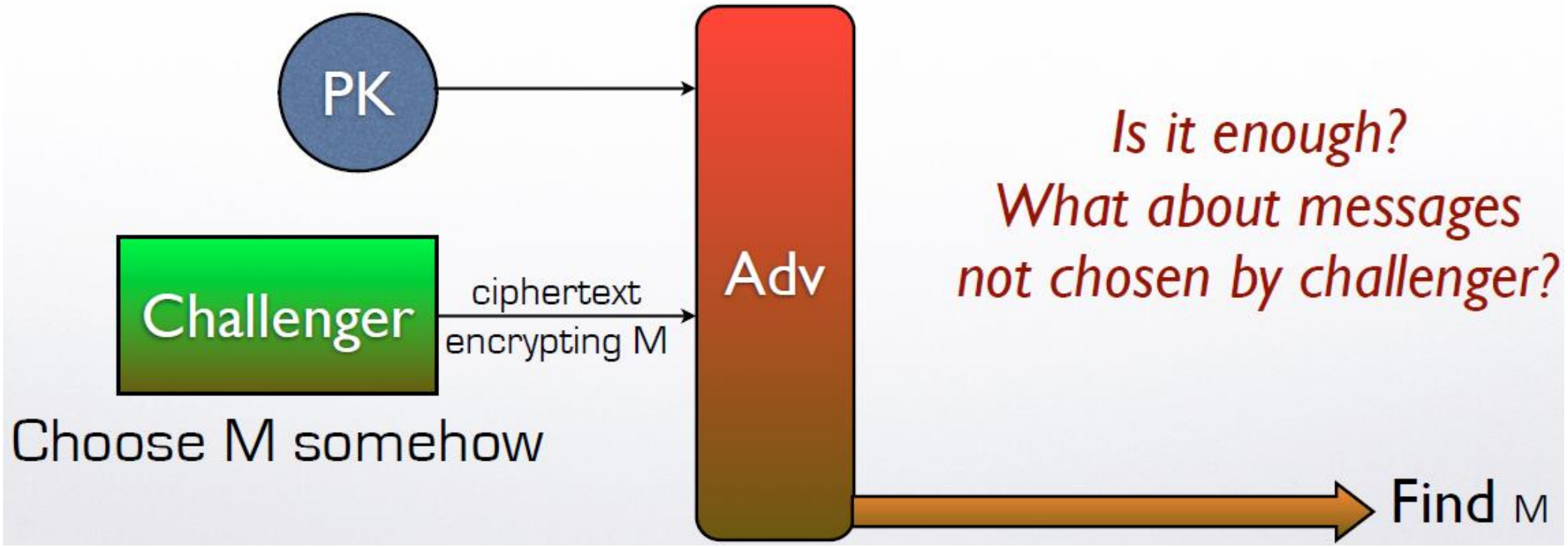
Public Key Encryption

- Three algorithms:
- $(pk, sk) \leftarrow Gen(1^k)$, where k is a security parameter and Gen is a probabilistic polynomial time (ppt) algorithm, i.e. polynomial in its input size (in this case polynomial in k since 1^k is represented by k bits)
- $C \leftarrow Enc_{pk}(M)$, where Enc is a ppt algorithm, and
- $M \leftarrow Dec_{sk}(C)$ with probability $\geq 1 - negl(k)$ for $C \leftarrow Enc_{pk}(M)$ with sk such that $(pk, sk) \leftarrow Gen(1^k)$, and where Dec is a ppt algorithm

- Public Key Encryption implies Key Agreement

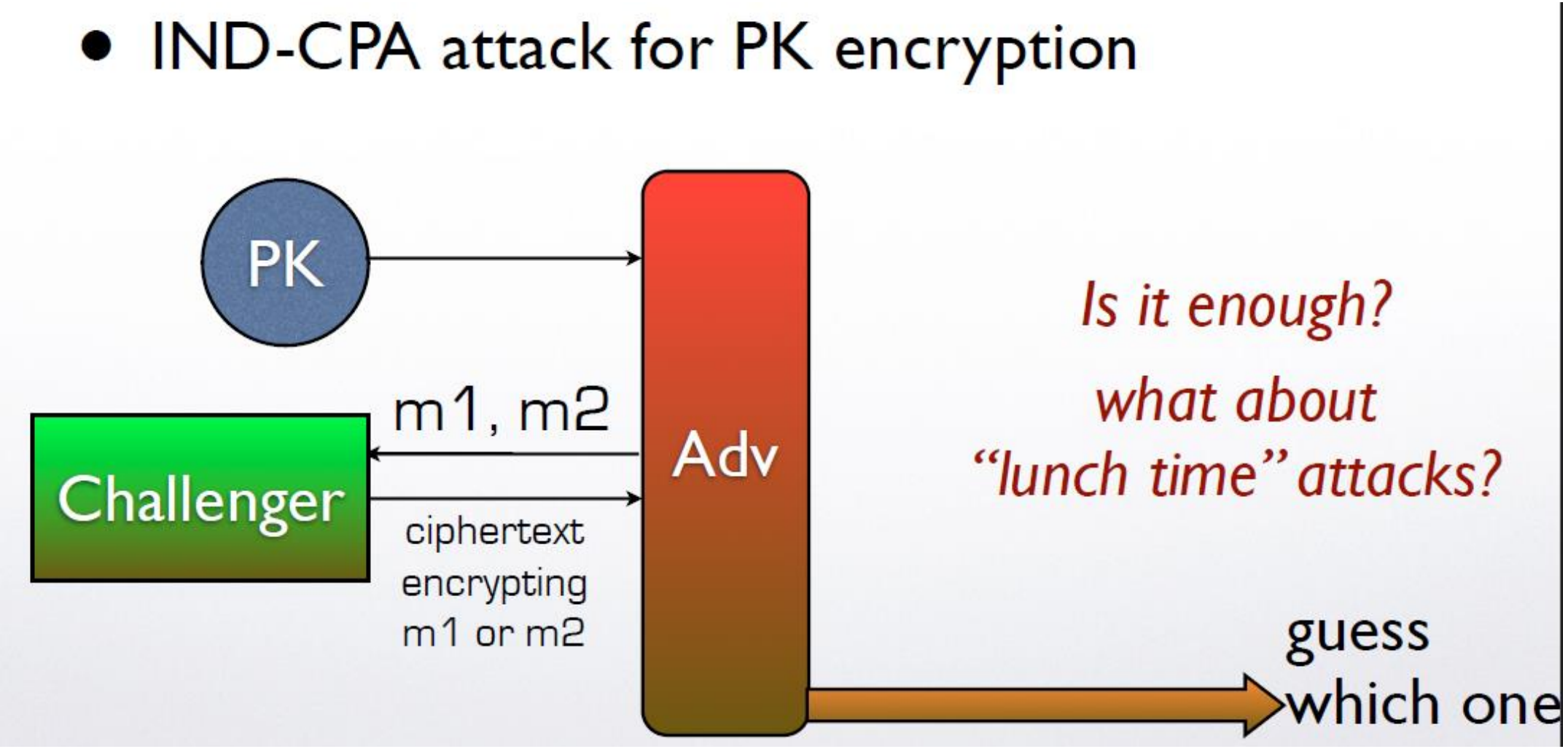
Modeling Security I

- Ciphertext-only attack for PK encryption



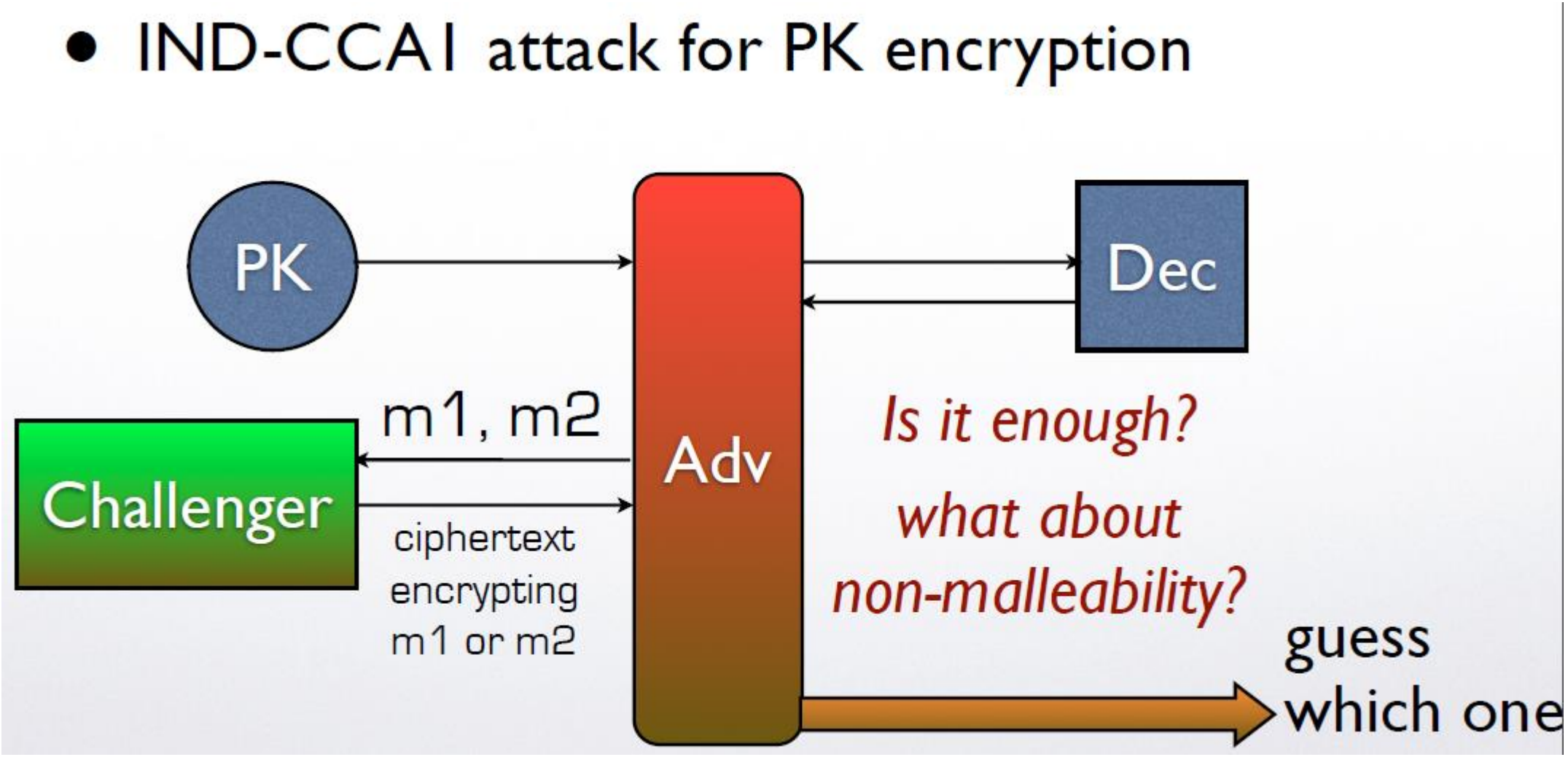
Modeling Security II

- IND-CPA attack for PK encryption



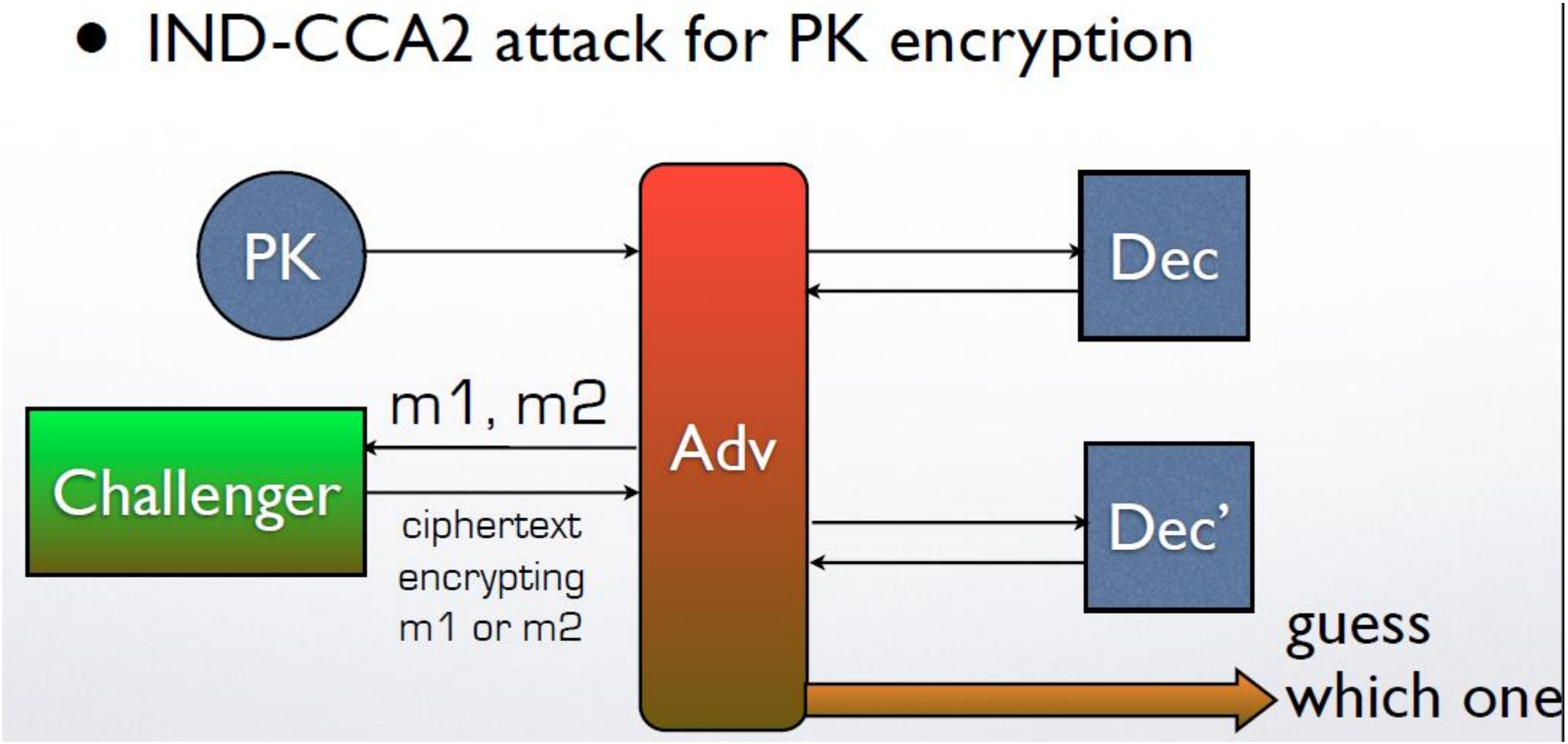
Modeling Security III

- IND-CCA1 attack for PK encryption



Modeling Security IV

- IND-CCA2 attack for PK encryption

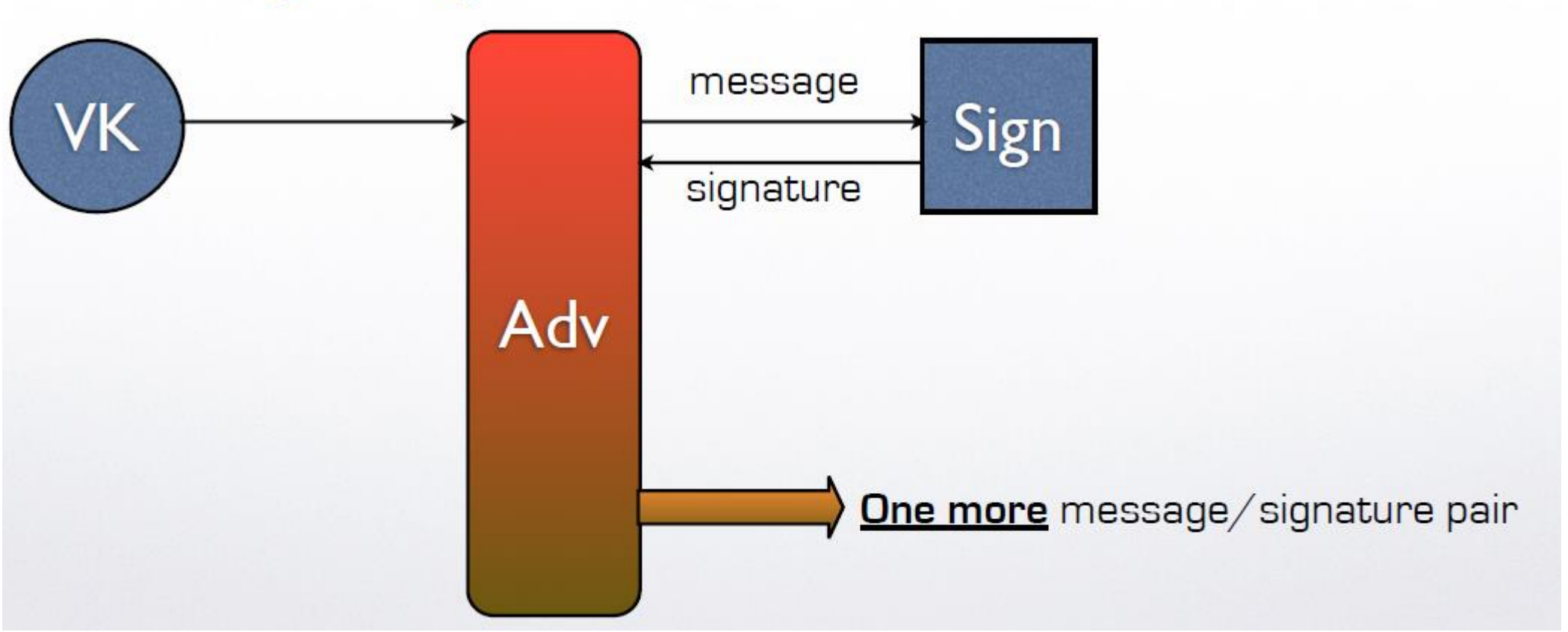


Digital Signatures

- We still want to authenticate the origin of a message or ciphertext
- Signature $y \leftarrow Sign_{sk}(M)$
- Verification $Yes\ or\ No \leftarrow Ver_{pk}(y)$ verifies whether message signature pair (M, y) comes from the owner of (someone who knows) the public secret key pair

Modeling Security

- Adaptive One-more Forgery Attack for Digital signatures



Crypto Background & Concepts

- Public Key Cryptography:
  - Key Agreement
  - Encryption
  - Signatures
  - Certificate Authority
  - Key Management and Public Key Infrastructure

- Hashes and MACs

HASH

- A means to produce a "fingerprint" or "log" or "measurement" of a file:
- $H \in \{0,1\}^* \rightarrow \{0,1\}^n$

- Properties
  - Efficiency
  - A good spread for various input distributions

- Allows a short representation of a file: Instead of M, use $H(M)$ as argument in a signature. Now $H(M)$ "commits" to M.

- Security?

Attacks with Collisions

- Collision attack: Find x and y such that H(x)=H(y)
- Second pre-image attack: Given x, find y such that H(x)=H(y)

Attacks against Secrecy

- When hashing is used to hide data:
  - Given H(M)=h, the goal is to recover M
  - Called “(first) pre-image attack”
  - Important for e.g. password hashing etc., generally important for hashing of secret (sensitive or critical) data

Birthday Paradox

- How many people should be in a room so that the probability that two of them share a birthday becomes larger than 50%?

$$\Pr[not\ collison\ among\ k\ people] = \frac{n}{n}\frac{n-1}{n}\frac{n-2}{n}\ldots\frac{n-k+1}{n} = \prod_{l=1}^{k}\left(1-\frac{l}{n}\right)$$
$$\leq e^{-\frac{1}{n}\sum_{l=1}^{k} l} = e^{-\frac{k(k+1)}{2n}}$$

implies $\Pr[Collision] = \frac{1}{2}$ if $k \approx 1.177\sqrt{n}$

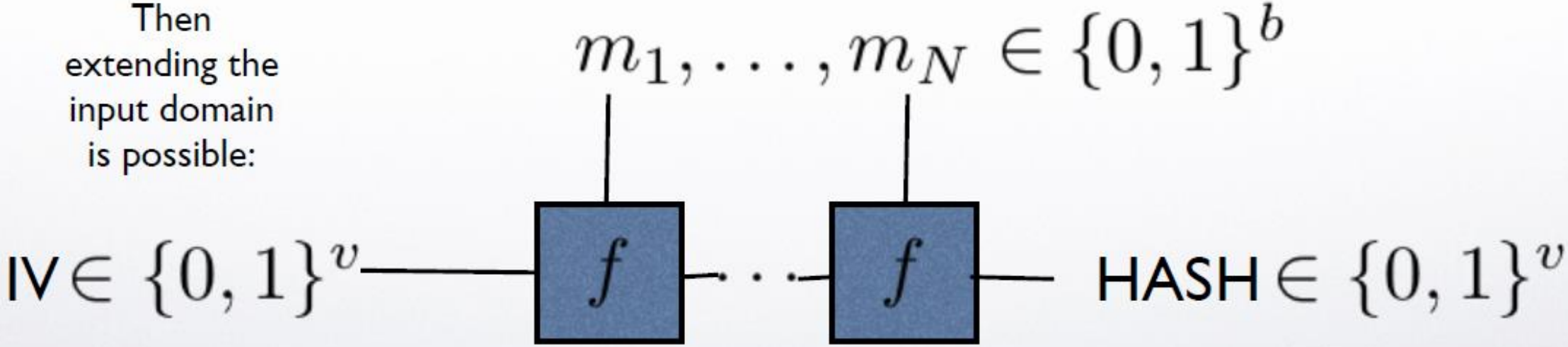
- Random sampling x, H(x)
- Store k pairs in a table
- Sort the table according to H(x)
- Linear pass to find if any entries have equal H(x)
- Storage k, Time ~2k+k log k, Choose k as above to obtain 50% probability of success
- The length of the hash function output is typically double the length of the key of an encryption algorithm

Merkle Damgard Design

- Suppose that you have a good "compression" function.

$$f : \{0,1\}^v \times \{0,1\}^b \rightarrow \{0,1\}^v$$

Then extending the input domain is possible:



Observation

If the previous construction is used then given $\mathcal{H}(m)$

One can easily find the hash of $m||m'$ How?

Hash functions constructed as above are called

Iterated Hash Functions

The SHA Family

- SHA-0 was made a standard by NIST in 1993. It was designed by NSA. Also based on Merkle Damgard design. 160 bits.
- In 1995 a technical revision was added that added an additional rotation operation. This was SHA-1.
- In 1998 collisions against SHA-0 were demonstrated in 2^{61} steps (Chabaud - Joux Crypto 1998)

The SHA Family

- SHA-1 was thought to be secure but evidence to the contrary was emerging (near-collisions, collisions on "reduced round" versions etc.)
- Finally: Collisions were found in 2^{69} steps Wang, Yin, Yu, Crypto 2005.
- SHA-1 is considered broken and its usage will be discontinued.

The SHA Family

- Isn't 2^{69} still too high?
- [Wang-Yao-Yao'05] announced new collision attacks of complexity: 2^{63}
- [De Caniere and Rechberger'06] - towards more structured collisions for SHA-1
- In 2009 claims for 2^52 were made (?).

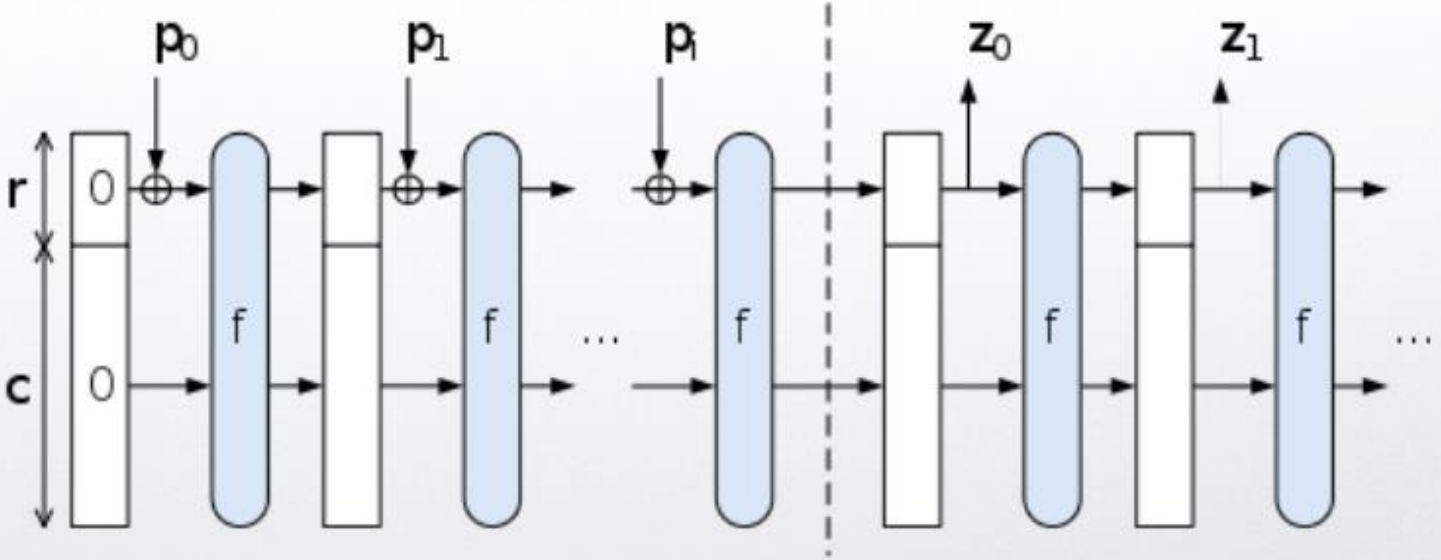
Nevertheless

- The above results do not necessarily mean that current products that use SHA-1 are insecure [remember our definition of security]. Moreover
  - There are ways to employ hash functions so that collision finding is made harder(e.g., "salting" techniques).
  - Modify hash functions in a modular way.

SHA-3

224, 256, 384 and 512-bit output

- started 2007 - a competition by NIST
- In 2012 Keccak was selected as the SHA-3 winner
  - Blake
  - Grostl
  - JH
  - Keccak
  - Skein



MACs: Message Authentication Codes

- Keyed hash functions: $\mathcal{H}_k : \{0,1\}^* \rightarrow \{0,1\}^n$

Required Property
Computational Resistance:

Given any sequence of pairs

$$\langle m_1, \mathcal{H}_k(m_1)\rangle, \ldots, \langle m_v, \mathcal{H}_k(m_v)\rangle$$

It is hard to produce an additional pair: $\langle m, \mathcal{H}_k(m)\rangle$

Usage of MACs

- Data Origin Authentication.
  - Sender and Receiver share the key *k*.
  - All messages have a MAC appended by Sender.
  - Receiver verifies the MACs (by recomputing them).

Constructing MACs

- Generic construction based on a hash function: $\mathcal{H}_k(m) = \mathcal{H}(k||m)$
- Not good: if an iterated hash function is employed this ***will allow an attack***.

a possible implementation:

$$\mathcal{H}_k(m) = \mathcal{H}(k||padding||m||k)$$

SGX Software Attestation

- Measurement
- Local Attestation
- Remote Attestation

Isolation & Attestation

- So far, we explained how Intel SGX creates and manages secure enclaves:
  - We analyzed in detail into what extend computation taking place in a secure enclave is vulnerable to privacy leakage through a breach of the secure enclave infrastructure itself
  - Hardware isolation turns out to be a powerful primitive
  - SGX also has a MEE (management encryption engine) which implements basic crypto primitives: encryption and memory integrity checking (next lecture)

- When a computation in a secure enclave produces output, we also need to be able to verify (attest) that the output indeed originates from the enclave
  - We will now explain how freshness and authenticity of results can be verified
  - Basic crypto primitives turn out to be a powerful primitive

Why Software Attestation?

- An enclave author creates an application module which should be executed in a secure way as it deals with sensitive information:
  - First the application module should be verified using proof carrying code etc. to show that the module itself does not have a bug which would e.g. leak sensitive info in unencrypted form in future reports etc.
  - Second, even if the application module is formally verified, the module should be executed in a secure environment which is trusted to resist strong adversaries … We want to inherit the security posture as promised by Intel SGX
- During the application module's computation, the enclave wants to report output to some remote party
- How can the remote party trust the reported results to come from the enclave?
  - The main idea is to challenge the enclave with e.g. a random nonce, which the enclave includes in its report
  - The SGX HW signs the report with an attestation key
  - The remote party now verifies the signature with a public verification key – this public key comes from a trusted data base maintained (and certified) by Intel
  - The signature should convince the remote party that the enclave's code was executed in a secure enclave (with its resistance to certain attacks), hence the report can as such be trusted
    - Also the nonce allows to check for freshness, i.e., no replay attack is possible (replaying an old report with its signature)
- What does this signature attest to exactly? What does it imply?

The Report Signature

- A straightforward solution:
  - When an enclave is executing, it calls EREPORT
  - The SGX HW (micro code implementing EREPORT) takes over:
  - It signs the
    - Report (results of the computation)
    - The enclave's measurement (its identity)
    - The nonce talked about before as part of the report data (the enclave puts it in before calling EREPORT)
    - Use a secret key with corresponding public key certified by Intel and stored in its database
- The signature scheme is too complex to put in HW
  - Need a HW/SW co-design
  - The SW part needs to execute in a secure environment → A special signing (called quoting) enclave authored by Intel
  - Now we need the application enclave to communicate its report to the quoting enclave who needs to verify its authenticity → Need local attestation
    - This is a general feature: any caller enclave can communicate a report over an authentic channel to a target enclave
  - The remote party should verify that the trusted quoting enclave SW has produced the signature → This is remote attestation

The Attestation Key

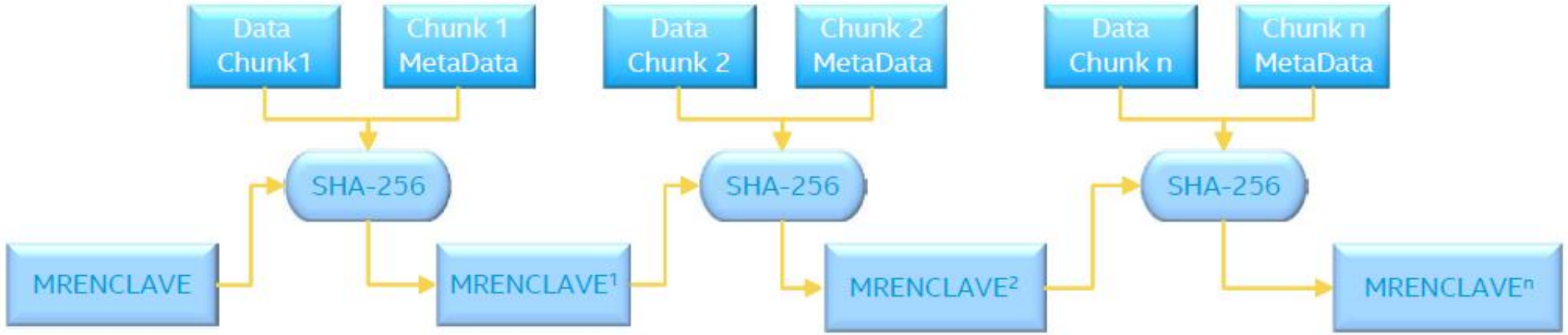
- Intel SGX does not fuse one static attestation key
- Multiple attestation keys can be associated with an Intel SGX processor
  - The processor may change ownership and for this reason a new attestation key may be issued (provisioned) by Intel
  - The signature scheme is such that ciphertexts cannot be linked through their attestation keys – unlinkability and anonymity
- Intel SGX fuses a provision key which is also stored by Intel
  - This provision key is used by a provisioning enclave to communicate with Intel to create an attestation key
  - This is another example of SW/HW co-design where this provisioning is done in SW (rather than dedicated HW)
- The attestation key need to be stored somewhere
  - Intel SGX fuses a sealing key – not known to anyone but the processor itself (the processor generates its own random key)
  - The sealing key is used by the provisioning enclave for creating a provisioning sealing key for authenticated encryption of the attestation key (which can only be decrypted by the quoting enclave)

Bootstrapping Trust

- We trust that the sealing key is only known by the SGX HW and the provisioning key is only known by the SGX HW and Intel's provisioning service
- These keys bootstrap trust in the generation of an attestation key with Intel's provisioning service and secure storage of the attestation key (by the Intel SGX processor)
- Intel's service is trusted as a third party and a signature verification key belonging to the attestation key can be retrieved by a client
- The application module in a caller enclave generates a report (with client nonce) and asks the SGX HW to MAC the report with a MAC key derived from the target enclave's report key and caller enclave's identity (measurement)
  - The SGX HW extracts the report key from the target enclave's identity (measurement) and sealing key
- The target enclave asks the SGX HW to produce its report key, and verifies the MAC after reconstruction of the MAC key
  - Note that only the SGX HW and target enclave can obtain the target enclave's report key
- If the target enclave is the quoting enclave, then the quoting enclave can extract the attestation key and produce an attestation signature
  - Only a proper quoting enclave could have obtained the attestation key
- The client verifies the signature with the verification key

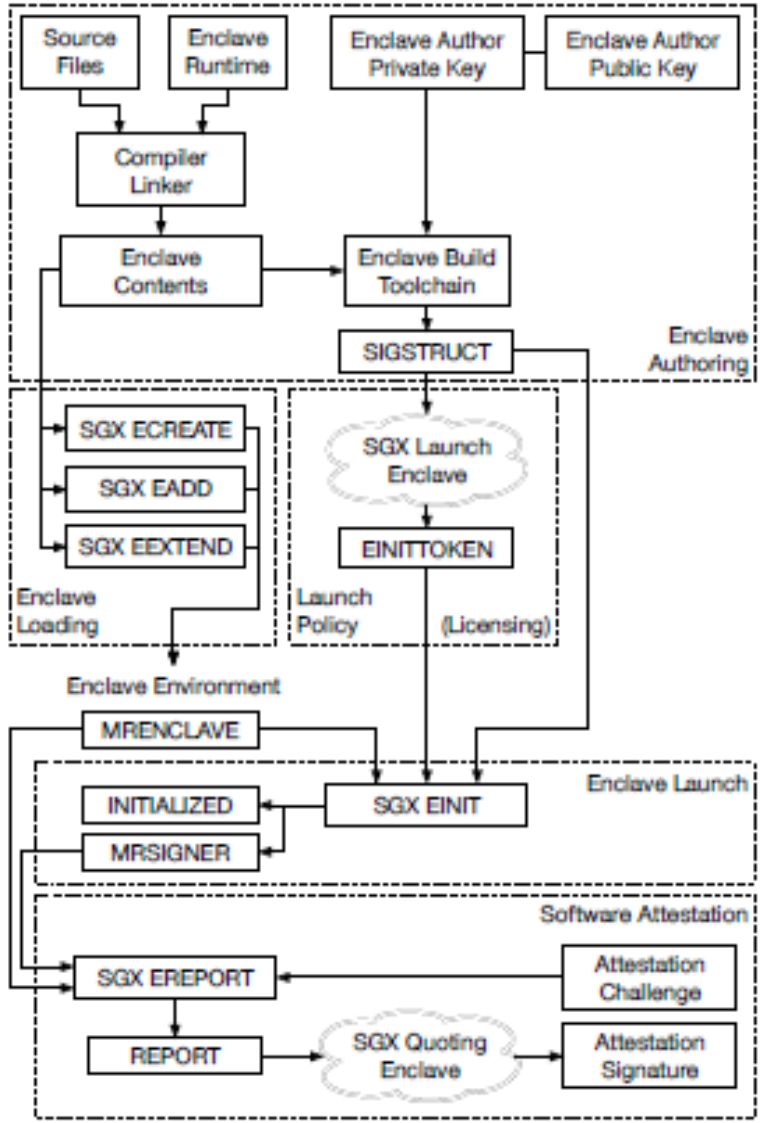
SGX Enclave Measurement (MRENCLAVE)

- When building an enclave, SGX generates a cryptographic log of all the build activities
  - Content: Code, Data, Stack, Heap
  - Location of each page within the enclave
  - Security flags being used
- MRENCLAVE ("Enclave Identity") is a 256-bit digest of the log
  - Represents the enclave's software TCB



Software Attestation

- Enclave author signs a measurement of the enclave content using its private key → SIGSTRUCT
- The enclave is created (see earlier lecture on its life cycle) and measured → MRENCLAVE
- A special enclave authored by Intel – the SGX Launch Enclave – is needed for remote attestation and produces EINITTOKEN
- A special enclave authored by Intel – the SGX Quoting Enclave – signs reports produced by EREPORT.
- This requires an attestation key which is provisioned by a Provisioning Enclave issued by Intel



Attestation

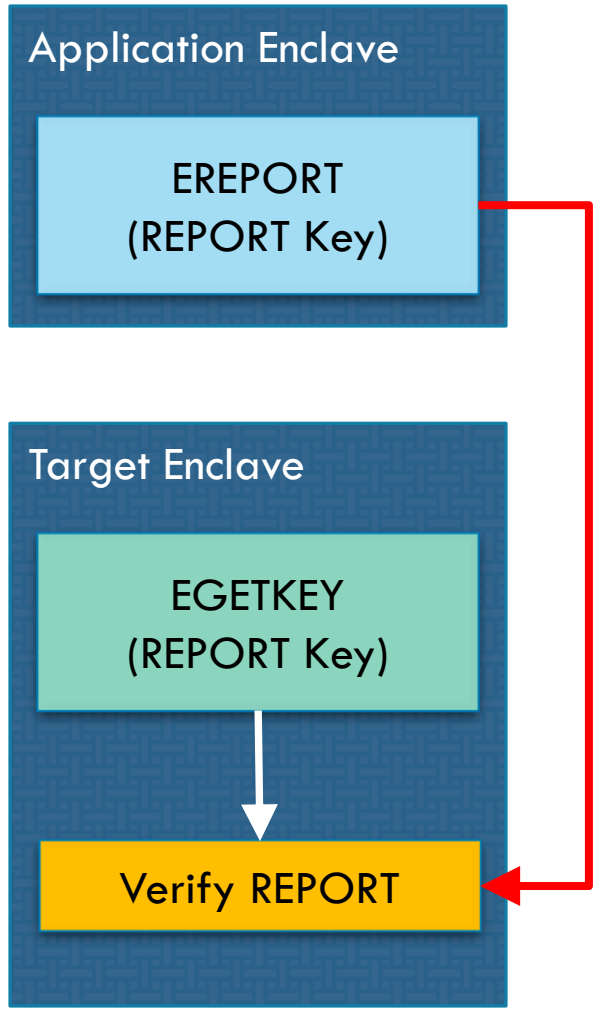
SGX provides LOCAL and REMOTE attestation capabilities

- ***Local attestation*** allows one enclave to attest its Thread Control Block (TCB) (i.e., the execution environment) to another enclave on the same platform
- ***Remote attestation*** allows one enclave to attest its TCB to another entity outside of the platform

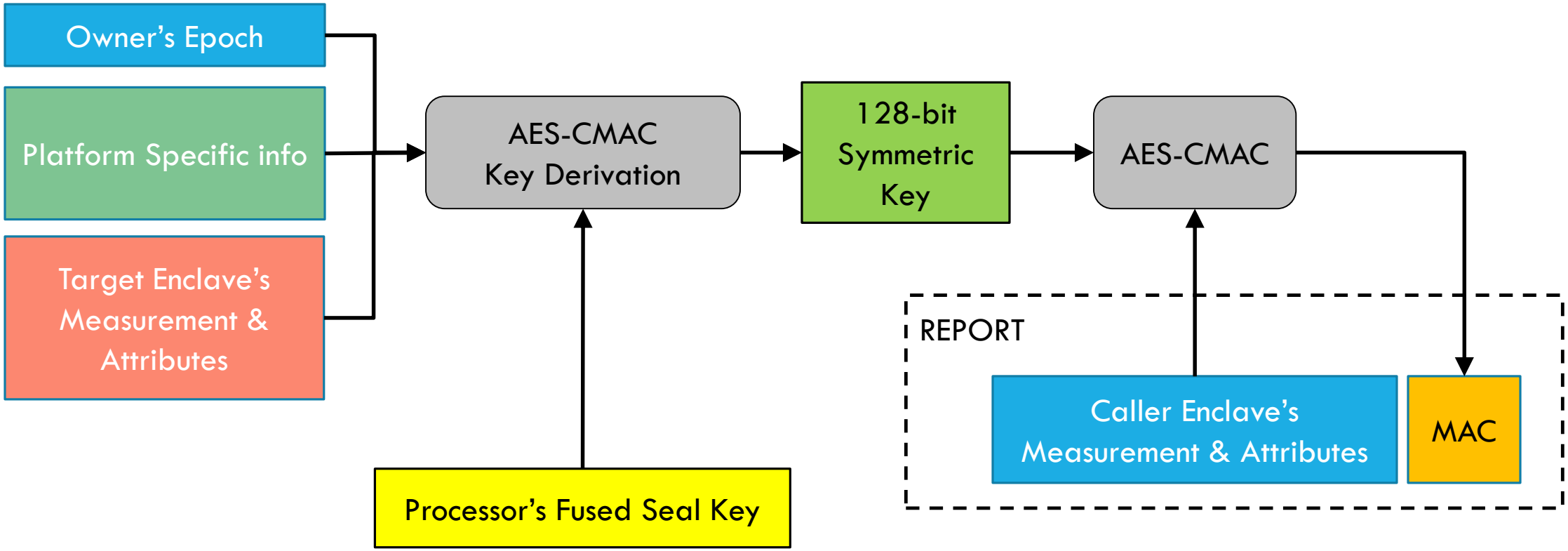
Local Attestation Overview

An application enclave proves its identity to another target enclave via the EREPORT instruction

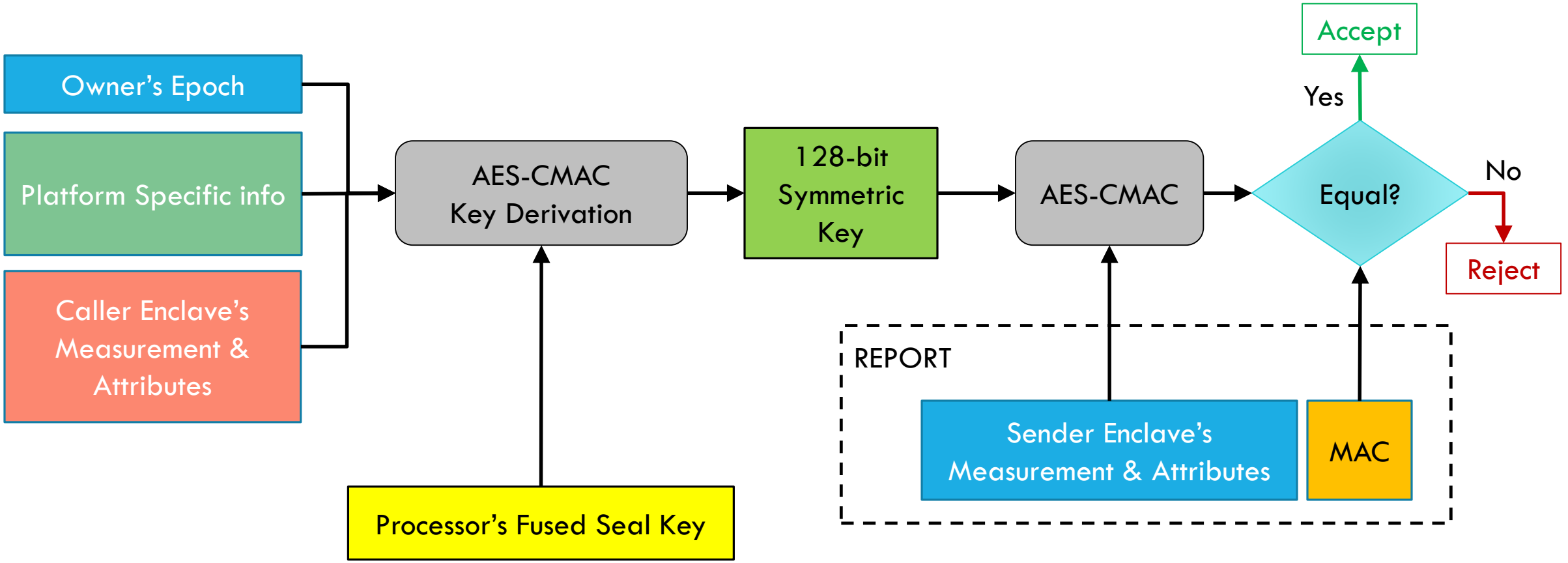
- Application Enclave calls EREPORT instruction to generate REPORT structure for a desired target enclave
  - REPORT contains calling enclave's Attributes, Measurements and User supplied data
- REPORT structure is secured using the REPORT key of the target enclave (shared only between the target enclave and SGX HW/microcode implementing EREPORT)
- EGETKEY is used by the target enclave to retrieve REPORT key
- Target enclave then verifies the REPORT structure using software (the report key together with an SGX master key, i.e. processor secret, and application enclave's measurement etc. is used to derive the MAC key)



REPORT Generation (EREPORT)

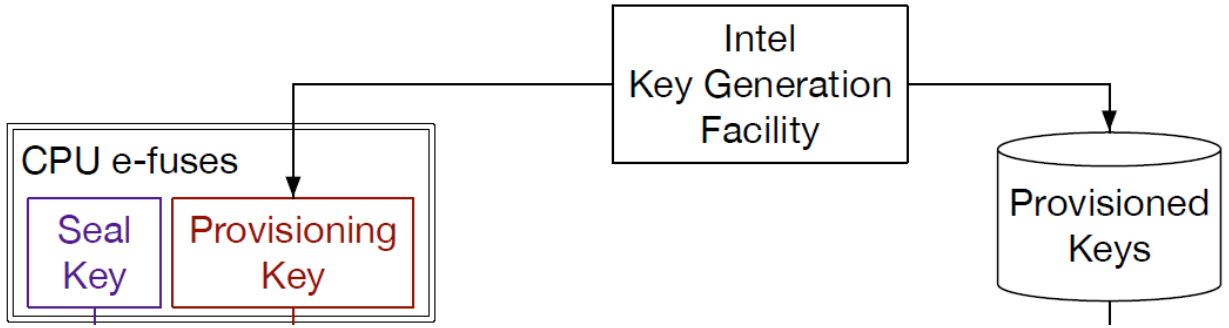


REPORT Verification (EGETKEY)



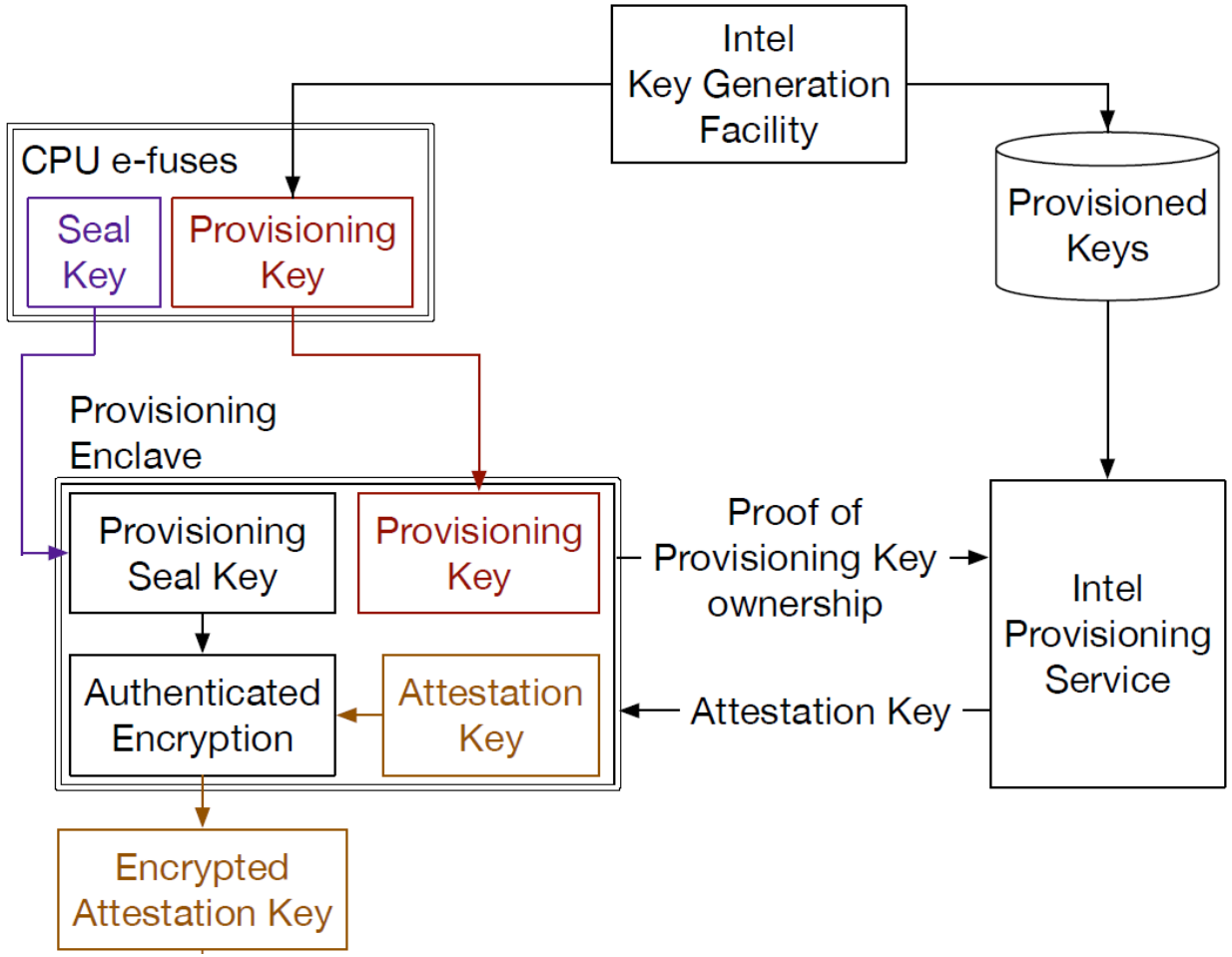
Remote Attestation

- During manufacturing, two keys are burned into the CPU
  - Fused Seal Key
  - Fused Provisioning Key
- Seal Key is used as Processor's secret and is generated inside the processor and not known to Intel
- Provisioning Key serves as a proof for a remote Platform and is also stored in a database at Intel



Remote Attestation

- Provisioning Key serves as a proof for remote Platform
- Remote platform issues an Attestation Key which is encrypted and stored for future use.



Remote Attestation

- A **Quoting Enclave** first performs Local Verification of application's Enclave
- Upon successful Local Attestation, Quoting Enclave decrypts the Attestation Key and signs the REPORT with this key
- The remote party verifies the signature (using the public part of the attestation key)