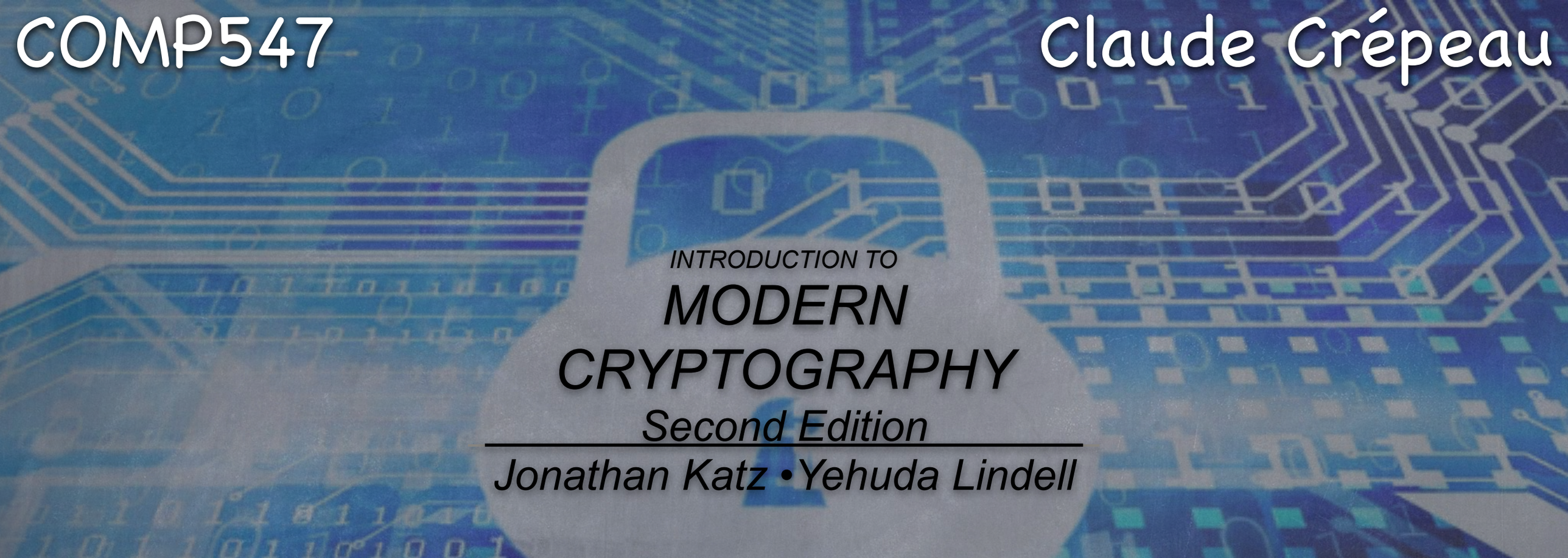
COMP547 Claude Crépeau



Chapter 12 : Digital Signature Schemes

Chapter 12 Digital Signature Schemes

Apologies: all numbering still refers to first edition of book.

12.1 Digital Signatures – An Overview

12.2 Definitions

12.3 RSA Signatures

12.3.1 "Plain **RSA**" and its Insecurity

12.3.2 Hashed **RSA**

12.4 The "Hash-and-Sign" Paradigm

12.5 Lamport's One-Time Signature Scheme

~~12.6 * Signatures from Collision-Resistant Hashing~~

~~12.6.1 "Chain-Based" Signatures~~

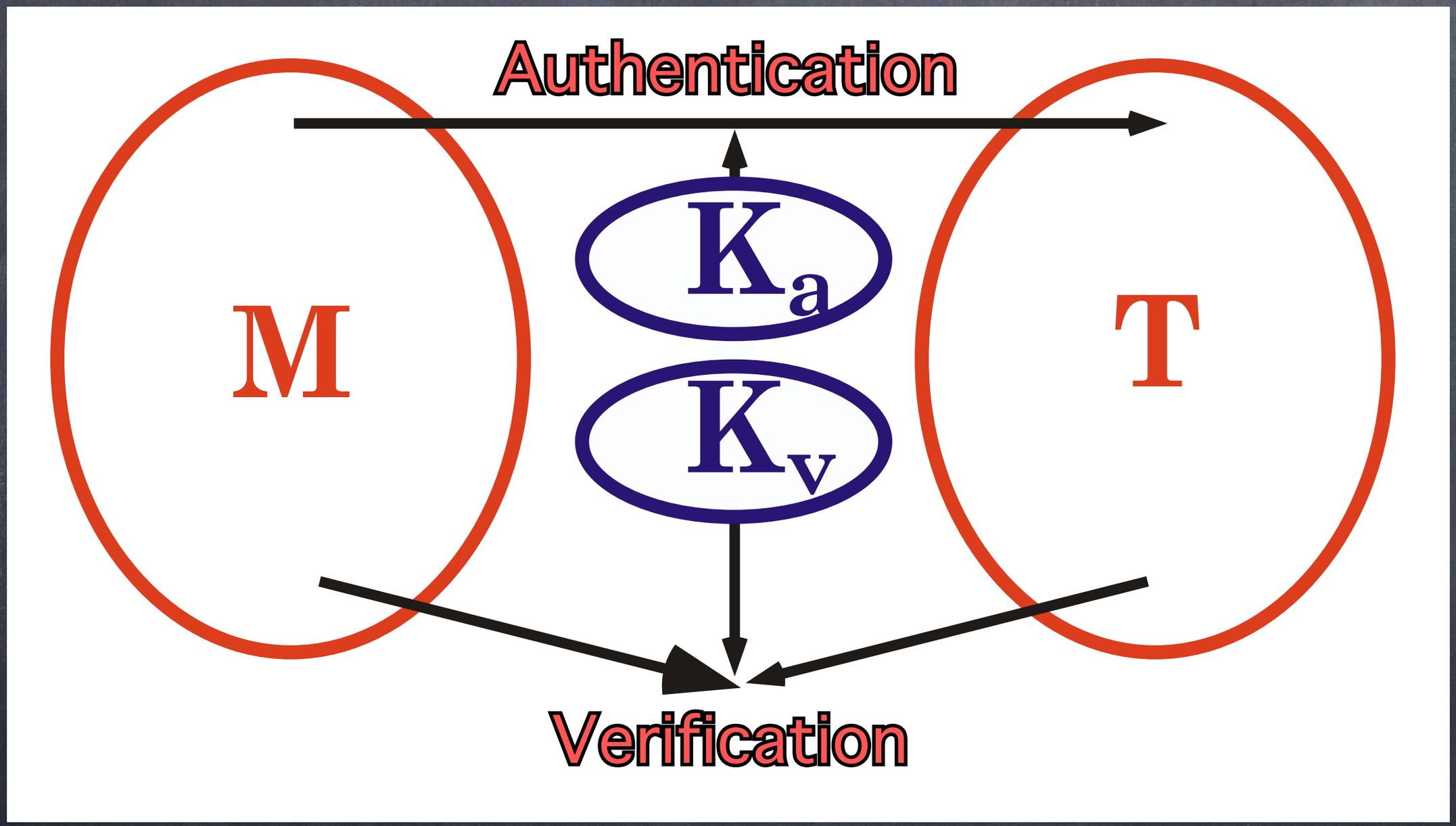
~~12.6.2 "Tree-Based" Signatures~~

12.7 The Digital Signature Standard (**DSS**)

12.8 Certificates and Public-Key Infrastructures

Asymmetric Authentication

(Digital Signature Scheme)



Complexity Theoretical Security

12.1 Digital Signatures – An Overview

Digital signature schemes allow a *signer* ***S*** who has established a public key ***pk***, to "*sign*" a message (using a secret key ***sk***) in such a way that any other party who knows ***pk*** (and knows that this public key was established by ***S***) can "*verify*" that the message originated from ***S*** and has not been modified in any way.

Digital Signatures – An Overview

As an example of typical usage of a digital signature scheme, consider a software company that wants to disseminate software patches in an authenticated manner:

that is, when the company needs to release a software patch it should be possible for any of its clients to recognize that the patch is authentic, and a malicious third party should never be able to fool a client into accepting a patch that was not actually released by the company.

Digital Signatures – An Overview

To do this, the company can generate a public key ***pk*** along with a private key ***sk***, and then distribute ***pk*** in some reliable manner to its clients while keeping ***sk*** secret.

Digital Signatures – An Overview

A malicious party might try to issue a fake patch by sending (m', σ') to a client, where m' represents a patch that was never released by the company.

This m' might be a modified version of some previous patch m, or it might be completely new and unrelated to previous patches. However, if the signature scheme is "*secure*", then when the client attempts to verify σ' it will find that this is an *invalid* signature on m' with respect to pk, and will therefore *reject* the signature.

Comparison to Message Authentication Codes

A *qualitative* advantage that digital signatures have as compared to message authentication codes is that signatures are *publicly verifiable*.

This means that if a receiver verifies the signature on a given message as being legitimate, then it is assured that all other parties who receive this signed message will also verify it as legitimate.

Comparison to Message Authentication Codes

- This feature is not achieved by message authentication codes where a signer shares a separate key with each receiver: in such a setting a malicious sender might compute a correct **MAC** tag with respect to receiver ***A***'s shared key but an incorrect **MAC** tag with respect to a different user ***B***'s shared key.

- In this case, ***A*** knows that he received an authentic message from the sender but has no guarantee that other recipients will agree.

Comparison to Message Authentication Codes

Public verifiability implies that signatures are *transferable* : a signature **σ** on a message ***m*** by a particular signer ***S*** can be shown to a third party, who can then verify herself that **σ** is a legitimate signature on ***m*** with respect to ***S***'s public key (here, we assume this third party also knows ***S***'s public key).

By making a copy of the signature, this third party can then show the signature to another party and convince *them* that ***S*** authenticated ***m***, and so on.

Transferability and public verifiability are essential for the application of digital signatures to certificates and public-key infrastructures.

Comparison to Message Authentication Codes

Digital signature schemes also provide the very important property of *non-repudiation*. That is — assuming a signer ***S*** widely publicizes his public key in the first place — once *S* signs a message he cannot later deny having done so.

This aspect of digital signatures is crucial for situations where a recipient needs to prove to a third party (say, a judge) that a signer did indeed "certify" a particular message (e.g., a contract): assuming ***S***'s public key is known to the judge, or is otherwise publicly available, a valid signature on a message is enough to convince the judge that ***S*** indeed signed this message.

Relation to Public-Key Encryption

- Digital signatures are often mistakenly viewed as the "inverse" of public-key encryption, with the roles of the sender and receiver interchanged.
- Historically, in fact, it has been suggested that digital signatures can be obtained by "reversing" public-key encryption, i.e., signing a message $\boldsymbol{m}$ by decrypting it to obtain σ, and verifying a signature σ by encrypting it and checking whether the result is $\boldsymbol{m}$.
- The suggestion to construct signature schemes in this way is (inspired by the RSA schemes) but *completely unfounded* : in most cases, it is simply inapplicable, and in cases when it is applicable it results in signature schemes that are completely insecure.

12.2 Definitions

DEFINITION 12.1 *A* signature scheme *is a tuple of three probabilistic polynomial-time algorithms* (Gen, Sign, Vrfy) *satisfying the following:*

1. *The* key-generation algorithm Gen *takes as input a security parameter* 1^n *and outputs a pair of keys* (pk, sk). *These are called the* public key *and the* private key, *respectively. We assume for convenience that* pk *and* sk *each have length at least* n, *and that* n *can be determined from* pk, sk.

2. *The* signing algorithm Sign *takes as input a private key* sk *and a message*[2] $m \in \{0,1\}^*$. *It outputs a signature* σ, *denoted as* $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$.

3. *The deterministic* verification algorithm Vrfy *takes as input a public key* pk, *a message* m, *and a signature* σ. *It outputs a bit* b, *with* $b = 1$ *meaning* valid *and* $b = 0$ *meaning* invalid. *We write this as* $b := \mathsf{Vrfy}_{pk}(m, \sigma)$.

Definitions

It is required that for every n, *every* (pk,sk) *output by* $\mathbf{Gen}(1^n)$, *and every* $m \in \{0,1\}^*$, *it holds that*

$$\mathbf{Vrfy}_{pk}(m,\mathbf{Sign}_{sk}(m)) = 1.$$

If $(\mathbf{Gen},\mathbf{Sign},\mathbf{Vrfy})$ *is such that for every* (pk,sk) *output by* $\mathbf{Gen}(1^n)$, *algorithm* $\mathbf{Sign}_{sk}$ *is only defined for messages* $m \in \{0,1\}^{\ell(n)}$, *then we say that* $(\mathbf{Gen},\mathbf{Sign},\mathbf{Vrfy})$ *is a* signature scheme for messages of length $\ell(n)$.

Definitions

• A signature scheme is used in the following way. One party ***S*** , who acts as the *sender*, runs **Gen(1^n)** to obtain keys **(*pk*,*sk*)**.

• The public key ***pk*** is then publicized as belonging to ***S*** ; e.g., ***S*** can put the public key on its webpage or place it in some public directory.

• As in the case of public-key encryption, we assume that any other party is able to obtain a legitimate copy of ***S***'s public key.

• When ***S*** wants to transmit a message ***m***, it computes the signature $\sigma \leftarrow \mathbf{Sign}_{sk}(m)$ and sends **(*m*,σ)**.

Definitions

- Upon receipt of (m,σ), a receiver who knows pk can verify the authenticity of m by checking whether $\mathbf{Vrfy}_{pk}(m,\sigma) \stackrel{?}{=} 1$.

- This establishes both that S sent m, and also that m was not modified in transit. As in the case of message authentication codes, however, it does not say anything about *when* m was sent, and replay attacks are still possible.

Security of signature schemes.

The signature experiment **Sig-forge$_{A,\Pi}(n)$**:

1. **Gen(1^n)** *is run to obtain keys* **(pk,sk)**.

2. Adversary ***A*** *is given* ***pk*** *and oracle access to* **Sign$_{sk}(\cdot)$**. *The adversary then outputs* **(m,σ)**.
Let **Q** *denote the set of messages whose signatures were requested by* ***A*** *during its execution.*

3. The output of the experiment is defined to be 1 *iff*

(1) Vrfy$_{pk}(m,\sigma) = 1$, *and* **(2) $m \notin Q$.**

Security of signature schemes.

DEFINITION 12.2 *A signature scheme* **Π = (Gen,Sign,Vrfy)** *is* existentially unforgeable under an adaptive chosen-message attack *if for all probabilistic polynomial-time adversaries* ***A****, there exists a negligible function* **negl** *such that:*

$$\mathbf{Pr[Sig\text{-}forge_{A,\Pi}(n) = 1] \leq negl(n)}.$$

10.3 RSA Signatures

CONSTRUCTION 12.3

Let GenRSA be as in the text. Define a signature scheme as follows:

- Gen: on input 1^n run GenRSA(1^n) to obtain (N, e, d). The public key is $\langle N, e \rangle$ and the private key is $\langle N, d \rangle$.
- Sign: on input a private key $sk = \langle N, d \rangle$ and a message $m \in \mathbb{Z}_N^*$, compute the signature

$$\sigma := [m^d \bmod N].$$

- Vrfy: on input a public key $pk = \langle N, e \rangle$, a message $m \in \mathbb{Z}_N^*$, and a signature $\sigma \in \mathbb{Z}_N^*$, output 1 if and only if

$$m \stackrel{?}{=} [\sigma^e \bmod N].$$

The "textbook RSA" signature scheme.

12.3.1 "Plain RSA" and its Insecurity

It is easy to see that verification of a legitimately generated signature is always successful since

$$\sigma^e = (m^d)^e = m^{[ed \bmod \varphi(N)]} = m^1 = m \bmod N.$$

The plain **RSA** signature scheme is insecure, however, as the following examples demonstrate.

A no-message attack

It is trivial to output a forgery for the plain **RSA** signature scheme *based on the public key alone*, without even obtaining any signatures from the legitimate signer.

Given a public key $\boldsymbol{pk} = \langle \boldsymbol{N}, \boldsymbol{e} \rangle$, choose an arbitrary $\boldsymbol{\sigma} \in \mathbb{Z}^*_{\boldsymbol{N}}$ and compute $\boldsymbol{m} := [\boldsymbol{\sigma}^{\boldsymbol{e}} \bmod \boldsymbol{N}]$.

Then output the forgery $(\boldsymbol{m}, \boldsymbol{\sigma})$. It is immediate that $\boldsymbol{\sigma}$ is a valid signature on $\boldsymbol{m}$, and this is obviously a forgery since no signature on $\boldsymbol{m}$ was generated by the owner of the public key.

Forging a signature on an arbitrary message

Say the adversary wants to forge a signature on the message $m \in \mathbb{Z}_N^*$ with respect to the public key $pk = \langle N, e \rangle$.

The adversary chooses a random $m_1 \in \mathbb{Z}_N^*$, sets $m_2 := [m/m_1 \bmod N]$, and then obtains signatures σ_1 and σ_2 on m_1 and m_2, respectively.

We claim that $\sigma := [\sigma_1 \cdot \sigma_2 \bmod N]$ is a valid signature on m. This is because $\sigma^e = (\sigma_1 \cdot \sigma_2)^e = (m_1^d \cdot m_2^d)^e = m_1^{ed} \cdot m_2^{ed} = m_1 m_2 = m \bmod N$, using the fact that σ_1, σ_2 are valid signatures on m_1, m_2.

12.3.2 Hashed RSA

The basic idea is to apply some function ***H*** to the message before signing it.

That is, the public and private keys are the same as before except that a description of some function $H : \{0,1\}^* \rightarrow \mathbb{Z}_N^*$ is now included as part of the public key.

A message $m \in \{0,1\}^*$ is signed by computing

$$\sigma := [H(m)^d \bmod N].$$

Verification of the pair (m,σ) is carried out by checking whether $\sigma^e \stackrel{?}{=} H(m) \bmod N$.

Hashed RSA

An immediate observation is that a minimal requirement for the above scheme to be secure is that H must be collision-resistant (see Section **4.6**): if it is not, and an adversary can find two different messages m_1, m_2 with $H(m_1) = H(m_2)$, then forgery is trivial.

(Note, however, that H need not be compressing.)

Since H must be a collision-resistant hash function, this modified scheme described is sometimes called the *hashed **RSA** signature scheme*.

Hashed RSA

- The no-message attack. The natural way to attempt the no-message attack shown previously is to choose an arbitrary $\sigma \in \mathbb{Z}_N^*$, compute $m' := [\sigma^e \bmod N]$, and then try to find some $m \in \{0,1\}^*$ such that $H(m) = m'$.

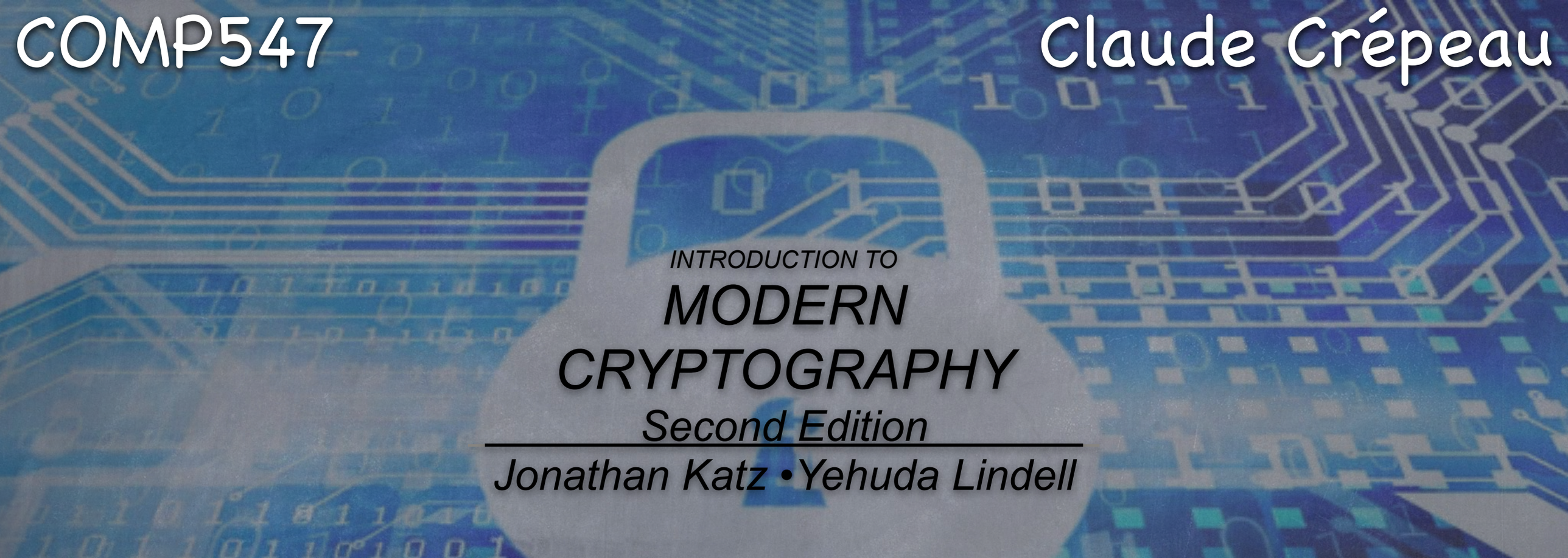
- If the function H is not efficiently invertible this appears difficult to do.

Hashed RSA

- Forging a signature on an arbitrary message. The natural way to attempt the chosen-message attack shown previously requires the adversary to find three messages m, m_1, m_2 for which

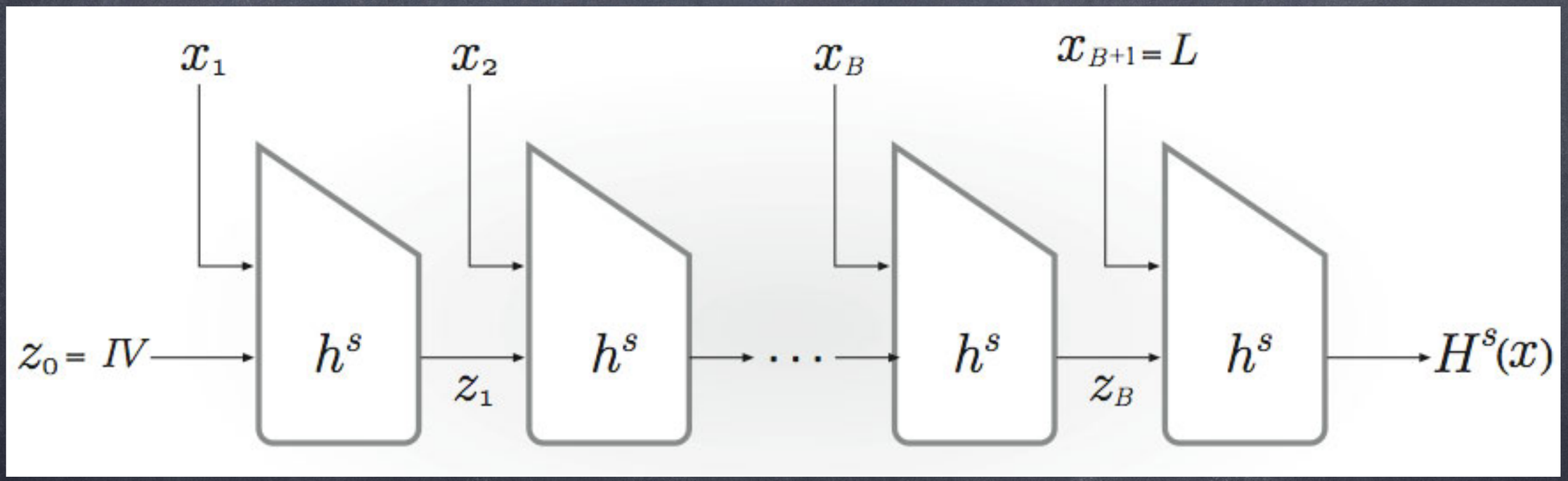
$$H(m) = [H(m_1) \cdot H(m_2) \bmod N\ ].$$

Once again, if H is not efficiently invertible this seems difficult to do.



Chapter 5 : Hash Functions and Applications

4.6 Collision-Resistant Hash Functions



4.6.1 Defining Collision Resistance

DEFINITION 4.11 *A* hash function *is a pair of probabilistic polynomial-time algorithms* (Gen, H) *fulfilling the following:*

- Gen *is a probabilistic algorithm which takes as input a security parameter* 1^n *and outputs a key* s*. We assume that* 1^n *is implicit in* s*.*

- *There exists a polynomial* ℓ *such that* H *takes as input a key* s *and a string* $x \in \{0,1\}^*$ *and outputs a string* $H^s(x) \in \{0,1\}^{\ell(n)}$ *(where* n *is the value of the security parameter implicit in* s*).*

If H^s *is defined only for inputs* $x \in \{0,1\}^{\ell'(n)}$ *and* $\ell'(n) > \ell(n)$*, then we say that* (Gen, H) *is a* fixed-length *hash function for inputs of length* $\ell'(n)$*.*

Defining Collision Resistance

The collision-finding experiment **Hash-coll$_{A,\Pi}(n)$**:

1. A key **s** *is generated by running* **Gen(1^n)**.

2. The adversary **A** *is given* **s** *and outputs* **x,x'** . (*If* **Π** *is a fixed-length hash function for inputs of length* **$\ell'(n)$** *then we require* **$x,x' \in \{0,1\}^{\ell'(n)}$**.)

3. The output of the experiment is defined to be **1** *if and only if* **$x \neq x'$** *and* **$H^s(x) = H^s(x')$**. *In such a case we say that* **A** *has found a collision.*

Defining Collision Resistance

DEFINITION 4.12 *A hash function* **Π = (Gen,*H*)** *is* collision resistant *if for all probabilistic polynomial-time adversaries* ***A*** *there exists a negligible function* **negl** *such that*

$$\mathbf{Pr[Hash\text{-}coll}_{A,\Pi}(n) = 1] \leq \mathbf{negl}(n).$$

4.6.2 Weaker Notions of Secure Hash Functions

1. ***Collision resistance:*** This is the strongest notion and the one we have considered so far.

2. ***Second pre-image resistance:*** Informally speaking, a hash function is second pre-image resistant if given s and x it is infeasible for a probabilistic polynomial-time adversary to find $x' \neq x$ s.t. $H^s(x') = H^s(x)$.

3. ***Pre-image*** *resistance:* Informally, a hash function is pre-image resistant if given s and $y = H^s(x)$ (but not x itself) for a randomly-chosen x, it is infeasible for a **PPT** adversary to find a value x' s.t. $H^s(x') = y$.

4.6.3 A Generic "Birthday" Attack

Assume we are given a hash function $H : \{0,1\}^* \to \{0,1\}^\ell$. The attack works as follows: Choose q arbitrary distinct inputs $x_1,\ldots,x_q \in \{0,1\}^{2\ell}$, compute $y_i := H(x_i)$, and check whether any of the two y_i values are equal.

What is the probability that this algorithm finds a collision?

A Generic "Birthday" Attack

When $q = \Theta(2^{\ell/2})$, the probability of such a collision is roughly ½.

In the case of birthdays, it turns out that if there are **23** people in a room, the probability that two have the same birthday is greater than ½.

10.4.2 Attacks on Plain RSA

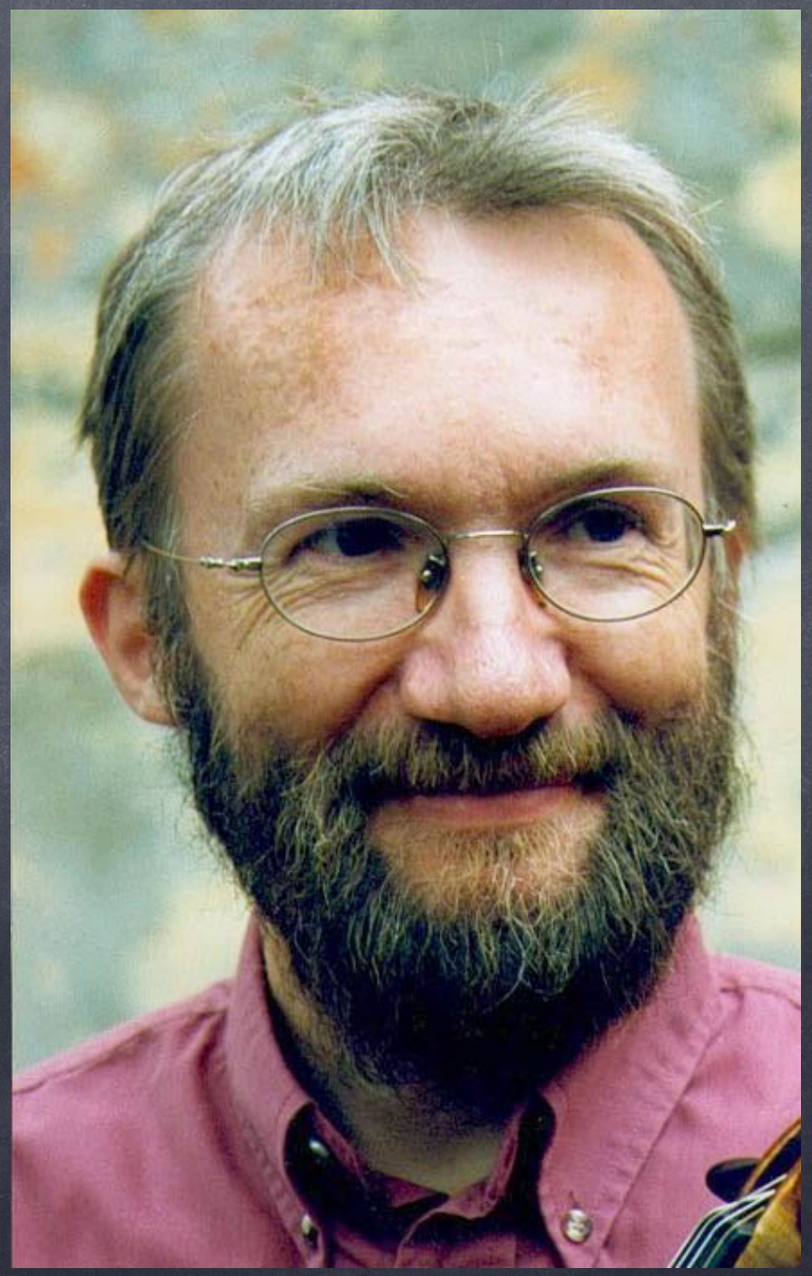
As an example, assume a hash function is designed with output length of **128** bits.

It is clearly infeasible to run 2^{128} steps in order to find a collision. However, running for 2^{64} steps is within the realm of feasibility (though still rather difficult).

Thus, the existence of generic birthday attacks mandates that any collision-resistant hash function in practice needs to have output that is longer than **128** bits.

4.6.4 The Merkle-Damgård Transform

Ralph Merkle



Ivan Damgård

4.6.4 The Merkle-Damgård Transform

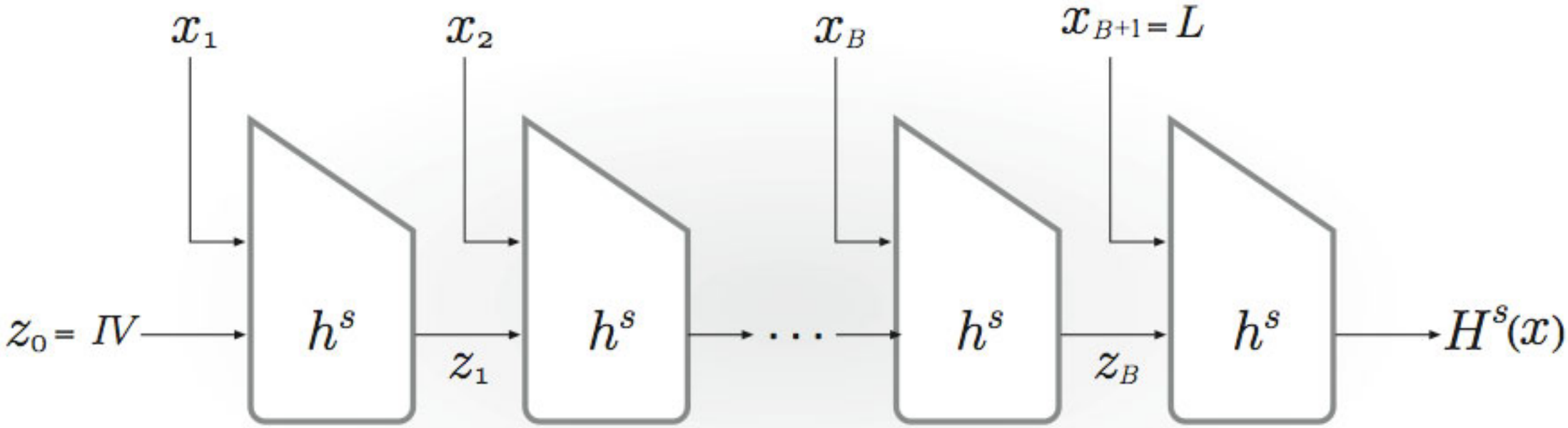


FIGURE 4.2: The Merkle-Damgård transform.

The Merkle-Damgård Transform

CONSTRUCTION 4.13

Let (Gen, h) be a fixed-length collision-resistant hash function for inputs of length $2\ell(n)$ and with output length $\ell(n)$. Construct a variable-length hash function (Gen, H) as follows:

- Gen: remains unchanged.
- H: on input a key s and a string $x \in \{0,1\}^*$ of length $L < 2^{\ell(n)}$, do the following (set $\ell = \ell(n)$ in what follows):
  1. Set $B := \lceil \frac{L}{\ell} \rceil$ (i.e., the number of blocks in x). Pad x with zeroes so its length is a multiple of ℓ. Parse the padded result as the sequence of ℓ-bit blocks $x_1, \ldots, x_B$. Set $x_{B+1} := L$, where L is encoded using exactly ℓ bits.
  2. Set $z_0 := 0^\ell$.
  3. For $i = 1, \ldots, B+1$, compute $z_i := h^s(z_{i-1} \| x_i)$.
  4. Output z_{B+1}.

The Merkle-Damgård transform.

The Merkle-Damgård Transform

The initialization vector.

The value z_0 used in step **2** of **Construction 4.13** is arbitrary and can be replaced by any *constant*.

This value is typically called the ***IV*** or *initialization vector*.

The Merkle-Damgård Transform

The security of the Merkle-Damgård transform.

The intuition behind the security of the Merkle-Damgård transform is that if two different strings $\boldsymbol{x}$ and $\boldsymbol{x'}$ collide in $\boldsymbol{H^s}$, then there must be *distinct* intermediate values $\boldsymbol{z_{i-1} \| x_i}$ and $\boldsymbol{z'_{i-1} \| x'_i}$ in the computation of $\boldsymbol{H^s(x)}$ and $\boldsymbol{H^s(x')}$, respectively, s. t. $\boldsymbol{h^s(z_{i-1} \| x_i) = h^s(z'_{i-1} \| x'_i)}$.

The Merkle-Damgård Transform

THEOREM 4.14 *If* **(Gen,*h*)** *is a fixed-length collision-resistant hash function, then* **(Gen,*H*)** *is a collision-resistant hash function.*

4.6.5 Collision-Resistant Hash Functions in Practice

Two popular hash functions are **MD5** and **SHA-1**. Both **MD5** and **SHA-1** first define a *compression function* that compresses fixed-length inputs by a relatively small amount (in our terms, this compression function is a *fixed-length* collision-resistant hash function).

Then the Merkle-Damgård transform (or something very similar) is applied to the compression function in order to obtain a collision-resistant hash function for arbitrary-length inputs.

Collision-Resistant Hash Functions in Practice

- The output length of **MD5** is **128** bits and that of **SHA-1** is **160** bits. The longer output length of **SHA-1** makes the generic "birthday attack" more difficult:

- for **MD5**, a birthday attack requires
  $\approx 2^{128/2} = 2^{64}$ hash computations,

- for **SHA-1** such an attack requires
  $\approx 2^{160/2} = 2^{80}$ hash computations.

Collision-Resistant Hash Functions in Practice

In **2004**, a team of Chinese cryptanalysts presented a breakthrough attack on **MD5** and a number of related hash functions.

Their technique for finding collisions gives little control over the collisions that are found; nevertheless, it was later shown that their method (and in fact any method that finds "random collisions") can be used to find collisions between, for example, two postscript files generating whatever viewable content is desired.

Collision-Resistant Hash Functions in Practice

- A year later, the Chinese team showed (theoretical) attacks on **SHA-1** that would find collisions using less time than that required by a generic birthday attack.
- The attack on **SHA-1** requires time $\mathbf{2^{69}}$ which lies outside the current range of feasibility; as of yet, no explicit collision in **SHA-1** has been found. (This is in contrast to the attack on **MD5**, which finds collisions in minutes.)

Collision-Resistant Hash Functions in Practice

These attacks have motivated a shift toward stronger hash functions with larger outputs lengths which are less susceptible to the known set of attacks on **MD5** and **SHA-1**.

Notable in this regard is the **SHA-2** family, which extends **SHA-1** and includes hash functions with **256**- and **512**-bit output lengths.

Another ramification of the attacks is that there is now great interest in designing new hash functions and developing a new hash standard.

Cryptographic Hash Project
Cryptographic Hash Algorithm Competition
Timeline for Hash Algorithm Competition
Federal Register Notices
Submission Requirements
Round 1
NEW! Round 2
Hash Forum
Contacts
Other Links

CRYPTOGRAPHIC HASH ALGORITHM COMPETITION

NIST has opened a public competition to develop a new cryptographic hash algorithm, which converts a variable length message into a short "message digest" that can be used for digital signatures, message authentication and other applications. The competition is NIST's response to recent advances in the cryptanalysis of hash functions. The new hash algorithm will be called "SHA-3" and will augment the hash algorithms currently specified in FIPS 180-2, Secure Hash Standard. Entries for the competition must be received by ***October 31, 2008***. The competition is announced in the Federal Register Notice published on November 2, 2007; further details of the competition will be available at the specific sites indicated in the menu on the left.

Cryptographic Hash Algorithm Competition

NIST noted some factors that figured into its selection as it announced the finalists:

- Performance: "A couple of algorithms were wounded or eliminated by very large [hardware gate] area requirement – it seemed that the area they required precluded their use in too much of the potential application space."
- Security: "We preferred to be conservative about security, and in some cases did not select algorithms with exceptional performance, largely because something about them made us 'nervous,' even though we knew of no clear attack against the full algorithm."
- Analysis: "NIST eliminated several algorithms because of the extent of their second-round tweaks or because of a relative lack of reported cryptanalysis – either tended to create the suspicion that the design might not yet be fully tested and mature."
- Diversity: The finalists included hashes based on different constructions, including the HAIFA and sponge hash constructions, and hashes with different sources of nonlinearity, including S-boxes and the interaction between addition and XOR.

NIST has released a report explaining its evaluation algorithm-by-algorithm.

Cryptographic Hash Algorithm Competition

NIST selected 51 entries for the Round 1.
14 of them advanced to Round 2,
from which 5 finalists were selected.

Finalists

NIST has selected five SHA-3 candidate algorithms to advance to the third (and final) round :

- BLAKE
- Grøstl (Knudsen et al.)
- JH
- Keccak (Keccak team, Daemen et al.)
- Skein (Schneier et al.)

NIST Selects Winner of Secure Hash Algorithm (SHA-3) Competition

From NIST *Tech Beat*: October 2, 2012

Contact: Chad Boutin
301-975-4261

The National Institute of Standards and Technology (NIST) today announced the winner of its five-year competition to select a new cryptographic hash algorithm, one of the fundamental tools of modern information security.

The winning algorithm, Keccak (pronounced "catch-ack"), was created by Guido Bertoni, Joan Daemen and Gilles Van Assche of STMicroelectronics and Michaël Peeters of NXP Semiconductors. The team's entry beat out 63 other submissions that NIST received after its open call for candidate algorithms in 2007, when it was thought that SHA-2, the standard secure hash algorithm, might be threatened. Keccak will now become NIST's SHA-3 hash algorithm.

Hash algorithms are used widely for cryptographic applications that ensure the authenticity of digital documents, such as digital signatures and message authentication codes. These algorithms take an electronic file and generate a short "digest," a sort of digital fingerprint of the content. A good hash algorithm has a few vital characteristics. Any change in the original message, however small, must cause a change in the digest, and for any given file and digest, it must be infeasible for a forger to create a different file with the same digest.

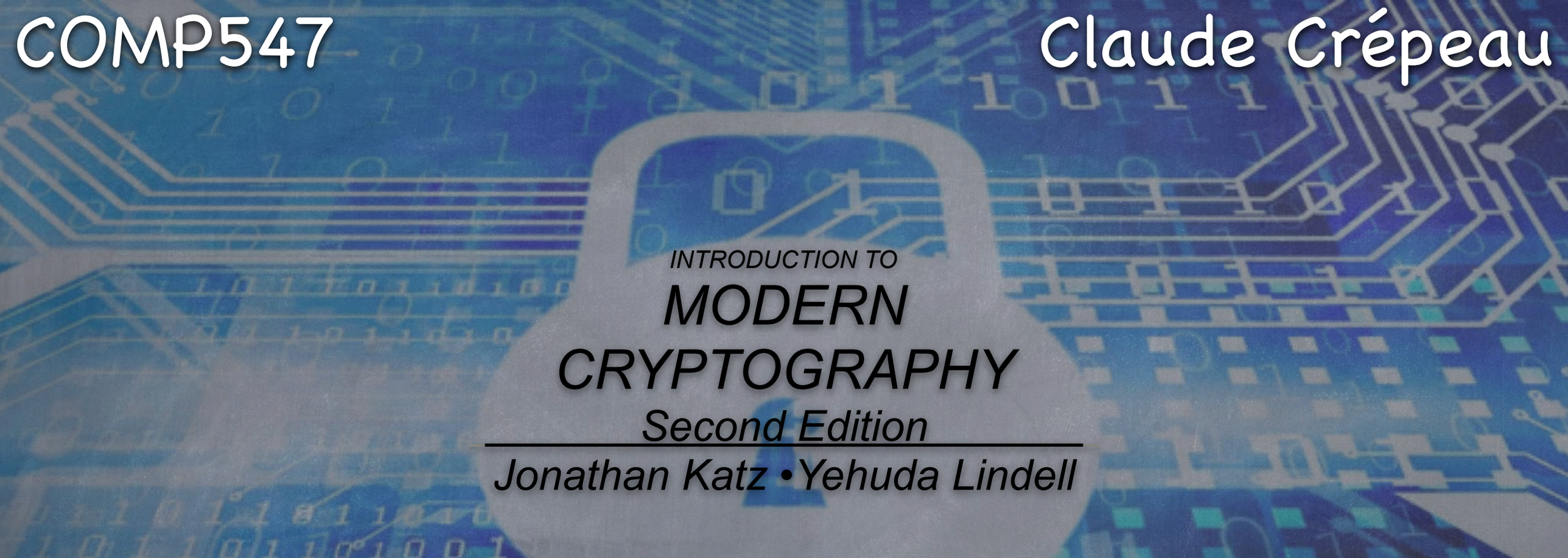
The NIST team praised the Keccak algorithm for its many admirable qualities, including its elegant design and its ability to run well on many different computing devices. The clarity of Keccak's construction lends itself to easy analysis (during the competition all submitted algorithms were made available for public examination and criticism), and Keccak has higher performance in hardware implementations than SHA-2 or any of the other finalists.

"Keccak has the added advantage of not being vulnerable in the same ways SHA-2 might be," says NIST computer security expert Tim Polk. "An attack that could work on SHA-2 most likely would not work on Keccak because the two algorithms are designed so differently."

Polk says that the two algorithms will offer security designers more flexibility. Despite the

Credit: K. Talbott/NIST with Shutterstock images
View hi-resolution image

COMP547 Claude Crépeau



Chapter 12 : Digital Signature Schemes

12.4 The "Hash-and-Sign" Paradigm

CONSTRUCTION 12.4

Let $\Pi = (\mathsf{Gen}_S, \mathsf{Sign}, \mathsf{Vrfy})$ and $\Pi_H = (\mathsf{Gen}_H, H)$ be as in the text. Construct a signature scheme Π' for arbitrary-length messages as follows:

- Gen': on input 1^n, run $\mathsf{Gen}_S(1^n)$ to obtain (pk, sk), and run $\mathsf{Gen}_H(1^n)$ to obtain s. The public key is $pk' = \langle pk, s \rangle$ and the private key is $sk' = \langle sk, s \rangle$.
- Sign': on input a private key $\langle sk, s \rangle$ and a message $m \in \{0,1\}^*$, compute $\sigma' \leftarrow \mathsf{Sign}_{sk}(H^s(m))$.
- Vrfy': on input a public key $\langle pk, s \rangle$, a message $m \in \{0,1\}^*$, and a signature σ, output 1 if and only if $\mathsf{Vrfy}_{pk}(H^s(m), \sigma) \stackrel{?}{=} 1$.

The hash-and-sign paradigm.

The "Hash-and-Sign" Paradigm

THEOREM 12.5 *If* Π *is existentially unforgeable under an adaptive chosen-message attack and* Π_H *is collision resistant, then* ***Construction 12.4*** *is existentially unforgeable under an adaptive chosen-message attack.*

12.7 The Digital Signature Standard(DSS)

CONSTRUCTION 12.15

Let $\mathcal{G}$ be as in the text. Define a signature scheme as follows:

- Gen: on input 1^n, run the algorithm $\mathcal{G}(1^n)$ to obtain (p, q, g). Let $H : \{0,1\}^* \to \mathbb{Z}_q$ be function. Choose $x \leftarrow \mathbb{Z}_q$ uniformly at random and set $y := [g^x \bmod p]$. The public key is $\langle H, p, q, g, y \rangle$ and the private key is $\langle H, p, q, g, x \rangle$.
- Sign: on input a private key $\langle H, p, q, g, x \rangle$ and a message $m \in \{0,1\}^*$, choose $k \leftarrow \mathbb{Z}_q^*$ uniformly at random and set $r := [\,[g^k \bmod p\,] \bmod q]$. Compute $s := [(H(m) + xr) \cdot k^{-1} \bmod q]$, and output the signature (r, s).
- Vrfy: on input a public key $\langle H, p, q, g, y \rangle$, a message $m \in \{0,1\}^*$, and a signature (r, s) with $r \in \mathbb{Z}_q$ and $s \in \mathbb{Z}_q^*$, compute the values $u_1 := [H(m) \cdot s^{-1} \bmod q]$ and $u_2 := [r \cdot s^{-1} \bmod q]$. Output 1 if and only if

$$r \stackrel{?}{=} [[g^{u_1} y^{u_2} \bmod p] \bmod q].$$

The Digital Signature Standard (DSS).

Digital Signature Standard

Another (vs **RSA**) important example is the *Digital Signature Standard (***DSS***)*, sometimes also known as the Digital Signature Algorithm (**DSA**).

This scheme was proposed by the National Institute of Standards and Technology (**NIST**) in **1991**, and has since become a **US** government standard.

The security of **DSS** relies on the hardness of the discrete logarithm problem, and has been used for many years without any serious attacks being found.

Digital Signature Standard

- However, there is no known proof of security for **DSS** based on the discrete logarithm (or any other) assumption.

- Moreover, **DSS** has no proof of security even in this idealized model.

- For these reasons, we must content ourselves with only giving a description of the scheme.

Digital Signature Standard

- Let G be a probabilistic polynomial-time algorithm that, on input 1^n, outputs (p,q,g) where, except with negligible probability:

(1) p and q are primes with $\|q\| = n$;

(2) $q \mid (p-1)$ but $q^2 \nmid (p-1)$; and

(3) g is a generator of the order q subgroup of $\mathbb{Z}_p^*$.

Digital Signature Standard

Let us see that the scheme is correct. Letting $m' = H(m)$, the signature (r, s) output by the signer satisfies

$$r = [\,[g^k \bmod p\,] \bmod q],$$
$$s = [(m' + xr) \cdot k^{-1} \bmod q]$$ and we check

$$r = [\,g^{u_1} y^{u_2} \bmod p\,] \bmod q$$ with
$$u_1 = m' \cdot s^{-1}$$
$$u_2 = r \cdot s^{-1}$$

Digital Signature Standard

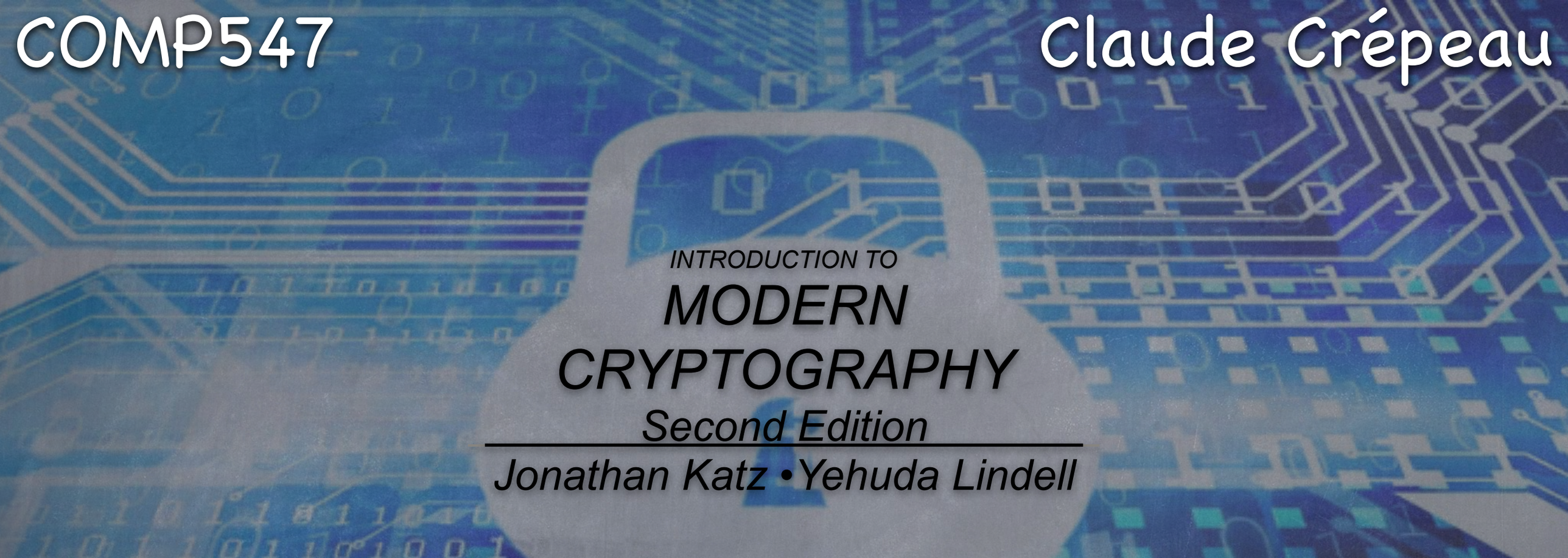
Assume $s \neq 0$ (this occurs with only negligible probability). Using the fact that $y = g^x$ and recalling that we can work "in the exponent" **modulo** q, we have

$$g^{u_1} y^{u_2} = g^{m's^{-1}} (g^x)^{rs^{-1}}$$
$$= g^{m'(m'+xr)^{-1}k} g^{xr\cdot(m'+xr)^{-1}k} \bmod p$$
Thus $$= g^{(m'+xr)\cdot(m'+xr)^{-1}k} \bmod p = g^k \bmod p.$$

$$[[g^{u_1} y^{u_2} \bmod p\ ] \bmod q]$$
$$= [[g^k \bmod p\ ] \bmod q] = r,$$

and verification succeeds.

COMP547 Claude Crépeau



Chapter 12 : Digital Signature Schemes

12.8 Certificates and P-K Infrastructures

- The key idea is the notion of a *digital certificate*, which is simply a signature binding some entity to some public key.

- To be concrete, say a party **Charlie** has generated a key-pair (pk_C, sk_C) for a secure digital signature scheme.

- Assume further that another party **Bob** has also generated a key-pair (pk_B, sk_B) (in the present discussion, these may be keys for either a signature scheme or a public-key encryption scheme), and that **Charlie** *knows* that pk_B is **Bob**'s public key.

Certificates and Public-Key Infrastructures

- Then **Charlie** can compute the signature
  $\mathbf{cert}_{C\to B} \stackrel{\text{def}}{=} \mathbf{Sign}_{sk_C}(\text{‘Bob’s key is } pk_B\text{ ’})$
  and give this signature to **Bob**.

- This signature $\mathbf{cert}_{C\to B}$ is called a *certificate* for **Bob**’s key issued by **Charlie**.

- In practice a certificate should unambiguously identify the party holding a particular public key and so a more uniquely descriptive term than “**Bob**” would be used, for example, **Bob**’s full name and email address.

Certificates and Public-Key Infrastructures

- Now say **Bob** wants to communicate with some other party **Alice** who already knows pk_C .
- What **Bob** can do is to send $(pk_B$, $\mathbf{cert}_{C \to B})$ to **Alice**, who can then verify the validity of the signature on the message '**Bob**'s key is pk_B' with respect to pk_C.
- Assuming verification succeeds, **Alice** now knows that **Charlie** has signed the indicated message.
- If **Alice** trusts **Charlie**, then she might now accept pk_B as **Bob**'s legitimate public key.

Certificates and Public-Key Infrastructures

- Note that all communication between **Bob** and **Alice** can occur over an *insecure* and *unauthenticated* channel.
- If an active adversary interferes with the communication of $(\mathbf{pk}_B, \mathbf{cert}_{C \to B})$ from **Bob** to **Alice**, that adversary will be unable to generate a valid certificate linking **Bob** to any *other* public key $\mathbf{pk}'_B$ unless **Charlie** had previously done so.
- This all assumes that **Charlie** is not dishonest and that his private signing key has not been compromised.

A single certificate authority

- The simplest **PKI** assumes a single *certificate authority* (**CA**) who is completely trusted by everybody and who issues certificates for everyone's public key.

- A certificate authority would not typically be a person, but would more likely be a company whose business it is to certify public keys, a governmental agency, or perhaps a department within an organization.

- Anyone who wants to rely on the services of the **CA** would have to obtain a legitimate copy of the **CA**'s public key pk_{CA}.

Multiple certificate authorities

- Outside of a single organization it is highly unlikely for *everyone* to trust the same **CA**.
- This need not imply that anyone thinks the **CA** is corrupt; it could simply be the case that someone finds the **CA**'s verification process to be insufficient.
- Moreover, the **CA** is a single point of failure for the entire system. If the **CA** is corrupt, or can be bribed, or even if the **CA** is merely lax with the way it protects its private signing key, the legitimacy of issued certificates may be called into question.

Multiple certificate authorities

Reliance on a single **CA** is also a problem even in non-adversarial environments:

• if the **CA** is unreachable then no new certificates can be issued, and

• the load on the **CA** may be very high if many parties want to obtain certificates at once (although this is still better than the **KDC** model because the **CA** does not need to be online once certificates are issued).

Multiple certificate authorities

- A party **Bob** who wants to obtain a certificate on his public key can choose which **CA**(s) it wants to issue a certificate, and a party **Alice** who is presented with a certificate, or even multiple certificates issued by different **CA**s, can choose which **CA**'s certificates she trusts.

- There is no harm in having **Bob** obtain a certificate from every **CA** (apart from some inconvenience and expense for **Bob**), but **Alice** must be more careful since the security of her communications is ultimately only as good as the least-secure **CA** that she trusts.

Delegation and certificate chains

- Say **Charlie**, acting as a **CA**, issues a certificate for **Bob** as in our original discussion.
- **Bob**, in turn, can issue his own certificates for other parties. For example, **Bob** may issue a certificate for **Alice** of the form

  $\mathbf{cert}_{B\to A} \stackrel{\text{def}}{=} \mathbf{Sign}_{sk_B}(\text{‘Alice’s key is } pk_A\text{’})$.

- Now, if **Alice** wants to communicate with some fourth party **Dave** who knows **Charlie**’s public key (but not **Bob**’s), then **Alice** can send

  pk_A, $\mathbf{cert}_{B\to A}$, pk_B, $\mathbf{cert}_{C\to B}$, to **Dave**.

Delegation and certificate chains

- **Dave** can first verify that **Charlie**, whom he trusts and whose public key is already in his possession, has signed a certificate $\mathbf{cert}_{C \to B}$ indicating that pk_B indeed belongs to someone named **Bob**.
- **Dave** can also verify that this person named **Bob** has signed a certificate $\mathbf{cert}_{B \to A}$ indicating that pk_A indeed belongs to **Alice**.
- If **Dave** trusts **Charlie** to only issue certificates to trustworthy people, then **Dave** may accept pk_A as being the authentic key of **Alice**.

The "web of trust" model

- In the "web of trust" model, anyone can issue certificates to anyone else and each user has to make their own decision about how much trust to place in certificates issued by other users.

- As an example of how this might work, say a user **Alice** is already in possession of public keys pk_1, pk_2, pk_3 for some users C_1, C_2, C_3.

- Another user **Bob** who wants to communicate with **Alice** might have certificates $cert_{C_1 \to B}$, $cert_{C_3 \to B}$, and $cert_{C_4 \to B}$, and will send these certificates (along with his public key pk_B) to **Alice**.

The "web of trust" model

- **Alice** cannot verify $\mathbf{cert}_{C_4 \to B}$ (she doesn't have C_4's public key), but she can verify the other two certificates
- Now she has to decide how much she trusts C_1 and C_3.
- She may decide to accept pk_B if she unequivocally trusts C_1, or if she trusts both C_1 and C_3 to a lesser extent.
- (She may, for example, consider it likely that either C_1 or C_3 is corrupt, but consider it unlikely for them *both* to be corrupt.)

Invalidating Certificates: Expiration

- One method for preventing certificates from being used indefinitely is to include an *expiry date* as part of the certificate.
- A certificate issued by a **CA** Charlie for **Bob**'s public key might now have the form

  $\mathbf{cert}_{C \to B} \stackrel{\text{def}}{=} \mathbf{Sign}_{sk_C}$('**Bob**'s key is pk_B', date),

  where "date" is some date in the future at which point the certificate becomes invalid. (For example, it may be one year from the day the certificate is issued.)

Invalidating Certificates: Expiration

- When another user verifies this certificate, they need to know not only pk_B but also the expiry date, and they now need to check not only that the signature is valid, but also that the expiry date has not passed.
- A user who holds a certificate must contact the **CA** to get a new certificate issued whenever their current one expires; at this point, the **CA** verifies the identity/credentials of the user again before issuing another certificate.

Invalidating Certificates: Revocation

- When an employee leaves an organization, or a user's private key is stolen, we would like the certificates that have been issued for their public keys to become invalid immediately, or at least as soon as possible.
- This can be achieved by having the **CA** explicitly *revoke* the certificate. Of course, everything we say applies more generally if the user had certificates issued by multiple **CAs**; for simplicity we assume a single **CA**.

Invalidating Certificates: Revocation

- There are many different ways revocation can be handled. One possibility is for the **CA** to include a serial number in every certificate it issues; that is, a certificate will now have the form

  $\mathbf{cert}_{C\to B}$ $\stackrel{\text{def}}{=}$ $\mathbf{Sign}_{sk_C}$(‘**Bob**’s key is $\boldsymbol{pk_B}$’,###),

  where “###” represents its serial number.

- Each certificate should have a unique serial number, and the **CA** will store the information **(Bob,**$\boldsymbol{pk_B}$**,###)** for each certificate it generates.

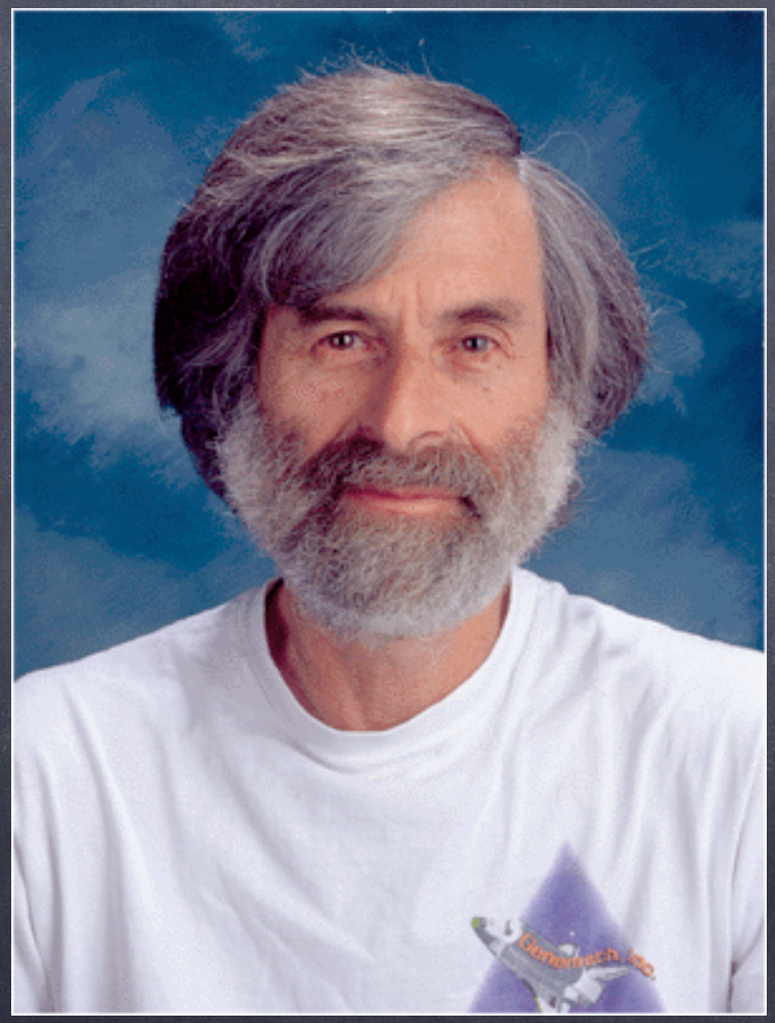
Invalidating Certificates: Revocation

- If a user **Bob**'s private key corresponding to the public key pk_B is stolen, then **Bob** can alert the **CA**.
- The **CA** will then search its database to find the serial number of the certificate issued for **Bob** and pk_B .
- At the end of each day, say, the **CA** will generate a *certificate revocation list* (or **CRL**) containing the serial numbers of all revoked certificates, and sign this entire list along with the current date. The signed list is then widely distributed, perhaps by posting it on the **CA**'s public webpage.

Invalidating Certificates: Revocation

- To verify a certificate issued as above, another user now needs pk_B and also the serial number of the certificate (this can be forwarded by Bob along with everything else).
- Verification now requires checking that the signature is valid, checking that the serial number does not appear on the most recent revocation list, and verifying the **CA**'s signature on the revocation list itself.

12.5 Lamport's One-Time Signature Scheme



Leslie Lamport

12.5 Lamport's One-Time Signature Scheme

Signing $m = 011$:

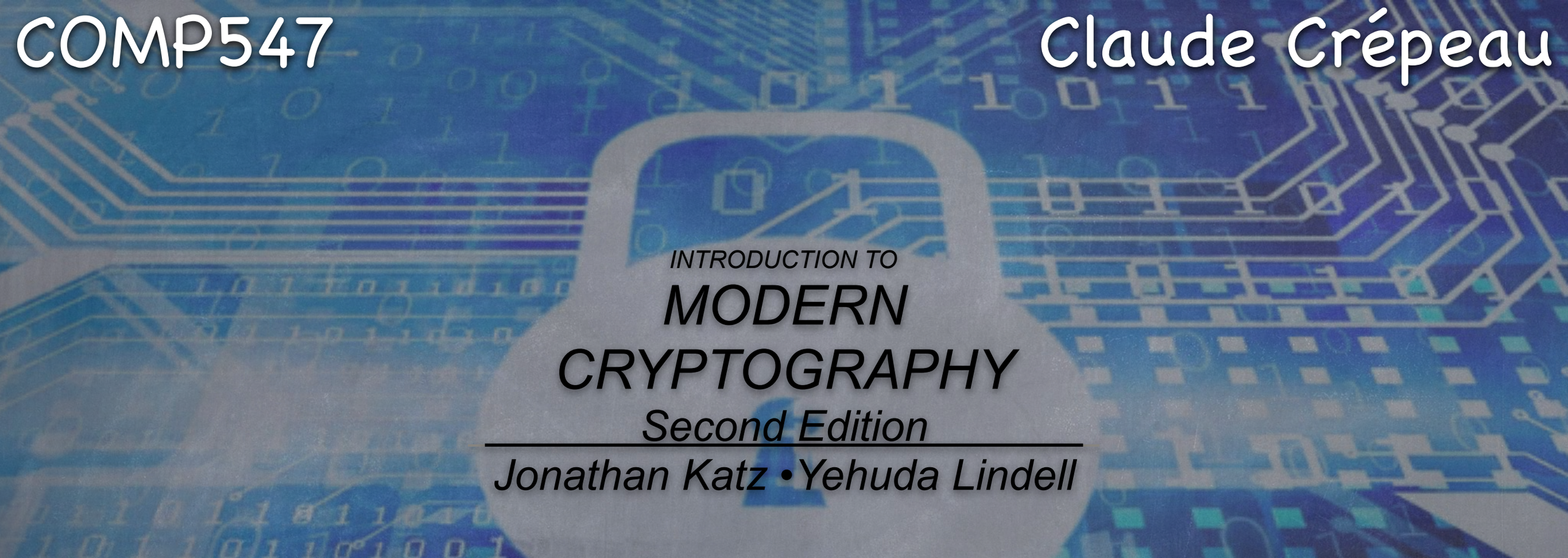
$$sk = \begin{pmatrix} \boxed{x_{1,0}} & x_{2,0} & x_{3,0} \\ x_{1,1} & \boxed{x_{2,1}} & \boxed{x_{3,1}} \end{pmatrix} \Rightarrow \sigma = (x_{1,0},\, x_{2,1},\, x_{3,1})$$

Verifying for $m = 011$ and $\sigma = (x_1, x_2, x_3)$:

$$pk = \left.\begin{pmatrix} \boxed{y_{1,0}} & y_{2,0} & y_{3,0} \\ y_{1,1} & \boxed{y_{2,1}} & \boxed{y_{3,1}} \end{pmatrix}\right\} \Rightarrow \begin{array}{l} f(x_1) \stackrel{?}{=} y_{1,0} \\ f(x_2) \stackrel{?}{=} y_{2,1} \\ f(x_3) \stackrel{?}{=} y_{3,1} \end{array}$$

FIGURE 12.1: The Lamport scheme used to sign the message $m = 011$.

COMP547 Claude Crépeau



Chapter 12 : Digital Signature Schemes