Radboud University

Public Key Encryption, Key Encapsulation Mechanisms, Digital Signatures

Applied Cryptography – Spring 2024

Simona Samardjiska

April 8, 2024

Institute for Computing and Information Sciences
Radboud University

Summary

Last time:

- Public Key Cryptography - a Recap
- Security of PKC
- Security of Public Key Encryption

Summary

Last time:

- Public Key Cryptography - a Recap
- Security of PKC
- Security of Public Key Encryption

Today:

- Security of Public Key Encryption (contd.)
- Key Encapsulation Mechanisms
- Digital Signatures from trapdoor permutations

Security of Public Key Encryption (PKE)

Baseline security: **indistinguishability under chosen-plaintext attacks (IND-CPA)**

A PKE scheme Π is called IND-CPA-secure if any PPT adversary $\mathcal{A}$ has only negligible advantage

$$Adv = \mathbf{Pr}\left(\mathrm{Exp}^{\text{ind-cpa}}_{\Pi(1^k)}(\mathcal{A}) = 1\right) - 1/2 = negl(k)\,.$$

in the following $\mathrm{Exp}^{\text{ind-cpa}}_{\Pi(1^k)}(\mathcal{A})$ **game (experiment)**:

Challenger		**Adversary**
$(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KGen}()$	$\xrightarrow{\mathrm{pk}}$	
	$\xleftarrow{M_i}$	M_i for number of i-s
	$\xrightarrow{\mathrm{Enc}(\mathrm{pk}, M_i)}$	
$b \xleftarrow{\$} \{0, 1\}$	$\xleftarrow{(M_0^*, M_1^*)}$	M_0^*, M_1^*
$C \leftarrow \mathrm{Enc}(\mathrm{pk}, M_b^*)$	$\xrightarrow{C}$	
	$\xleftarrow{b'}$	b'

Return 1 iff $b = b'$ otherwise 0.

An IND-CPA secure PKE - generic construction

An IND-CPA secure PKE - generic construction

Y computational problem (YC):

Let $S = \mathbb{Z}_p^*$ with generator g_1, and let $g_2 = g_1^s$. Let $T(x) = x^s$ be a trapdoor function.

Given: $\mathbb{Z}_p^*, g_1, g_2, g_1^a$

Find: g_2^a

An IND-CPA secure PKE - generic construction

Y computational problem (YC):

Let $S = \mathbb{Z}_p^*$ with generator g_1, and let $g_2 = g_1^s$. Let $T(x) = x^s$ be a trapdoor function.

Given: $\mathbb{Z}_p^*, g_1, g_2, g_1^a$

Find: g_2^a

Claim: YC is hard if CDH holds. **(Prove for homework by contradiction!)**

An IND-CPA secure PKE - generic construction

Y computational problem (YC):
Let $S = \mathbb{Z}_p^*$ with generator g_1, and let $g_2 = g_1^s$. Let $T(x) = x^s$ be a trapdoor function.
Given: $\mathbb{Z}_p^*, g_1, g_2, g_1^a$
Find: g_2^a

Claim: YC is hard if CDH holds. **(Prove for homework by contradiction!)**

- Remark: The YC problem can be defined much more general! (no need for it here)
- We further need a cryptographic hash function $G : S \to \{0,1\}^\ell$ modelled as a random oracle

An IND-CPA secure PKE - generic construction

Y computational problem (YC):
Let $S = \mathbb{Z}_p^*$ with generator g_1, and let $g_2 = g_1^s$. Let $T(x) = x^s$ be a trapdoor function.
 Given: $\mathbb{Z}_p^*, g_1, g_2, g_1^a$
 Find: g_2^a

Claim: YC is hard if CDH holds. **(Prove for homework by contradiction!)**

- Remark: The YC problem can be defined much more general! (no need for it here)
- We further need a cryptographic hash function $G : S \to \{0,1\}^\ell$ modelled as a random oracle

Construction of Π_0:
 KGen: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^k)$ where $g_2 = g_1^{\mathsf{sk}}$ and $\mathsf{pk} = (\mathbb{Z}_p^*, g_1, g_2)$. Further $T(x) = x^{\mathsf{sk}}$

An IND-CPA secure PKE - generic construction

Y computational problem (YC):
Let $S = \mathbb{Z}_p^*$ with generator g_1, and let $g_2 = g_1^s$. Let $T(x) = x^s$ be a trapdoor function.

Given: $\mathbb{Z}_p^*, g_1, g_2,\ g_1^a$

Find: g_2^a

Claim: YC is hard if CDH holds. **(Prove for homework by contradiction!)**

- Remark: The YC problem can be defined much more general! (no need for it here)
- We further need a cryptographic hash function $G : S \to \{0,1\}^\ell$ modelled as a random oracle

Construction of Π_0:

KGen: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^k)$ where $g_2 = g_1^{\mathsf{sk}}$ and $\mathsf{pk} = (\mathbb{Z}_p^*, g_1, g_2)$. Further $T(x) = x^{\mathsf{sk}}$

Enc: Choose $R \xleftarrow{\$} S$ and compute $\kappa = G(g_2^R)$

$$(C_1, C_2) \leftarrow \mathsf{Enc}(M, R) = (g_1^R, \kappa \oplus M)$$

An IND-CPA secure PKE - generic construction

Y computational problem (YC):

Let $S = \mathbb{Z}_p^*$ with generator g_1, and let $g_2 = g_1^s$. Let $T(x) = x^s$ be a trapdoor function.

Given: $\mathbb{Z}_p^*, g_1, g_2, g_1^a$

Find: g_2^a

Claim: YC is hard if CDH holds. **(Prove for homework by contradiction!)**

- Remark: The YC problem can be defined much more general! (no need for it here)
- We further need a cryptographic hash function $G : S \to \{0,1\}^\ell$ modelled as a random oracle

Construction of Π_0:

KGen: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^k)$ where $g_2 = g_1^{\mathsf{sk}}$ and $\mathsf{pk} = (\mathbb{Z}_p^*, g_1, g_2)$. Further $T(x) = x^{\mathsf{sk}}$

Enc: Choose $R \xleftarrow{\$} S$ and compute $\kappa = G(g_2^R)$

$$(C_1, C_2) \leftarrow \mathsf{Enc}(M, R) = (g_1^R, \kappa \oplus M)$$

Dec: Compute $\kappa = G(T(C_1))$ and output $M' \leftarrow \kappa \oplus C_2$

An IND-CPA secure PKE - generic construction

IND-CPA security: If the YC problem is hard then Π_0 is IND-CPA secure in the random oracle model (ROM).

An IND-CPA secure PKE - generic construction

IND-CPA security: If the YC problem is hard then Π_0 is IND-CPA secure in the random oracle model (ROM).

Sketch of proof: Suppose $\mathcal{A}$ has non-negligible advantage against Π_0 in an IND-CPA game (can (t, ϵ) break it). We construct simulator $\mathcal{B}$ that breaks the YC problem with non-negligible probability.

An IND-CPA secure PKE - generic construction

IND-CPA security: If the YC problem is hard then Π_0 is IND-CPA secure in the random oracle model (ROM).

Sketch of proof: Suppose $\mathcal{A}$ has non-negligible advantage against Π_0 in an IND-CPA game (can (t, ϵ) break it). We construct simulator $\mathcal{B}$ that breaks the YC problem with non-negligible probability.

Setup: $\mathcal{B}$ is given a YC instance $(\mathbb{Z}_p^*, g_1, g_2, g_1^a)$. His goal is to find g_2^a (but he does not know s).

$\mathcal{B}$ sets $\mathrm{pk} = (\mathbb{Z}_p^*, g_1, g_2)$ as the public key that $\mathcal{A}$ attacks in an IND-CPA game against Π_0.

An IND-CPA secure PKE - generic construction

IND-CPA security: If the YC problem is hard then Π_0 is IND-CPA secure in the random oracle model (ROM).

Sketch of proof: Suppose $\mathcal{A}$ has non-negligible advantage against Π_0 in an IND-CPA game (can (t, ϵ) break it). We construct simulator $\mathcal{B}$ that breaks the YC problem with non-negligible probability.

Setup: $\mathcal{B}$ is given a YC instance $(\mathbb{Z}_p^*, g_1, g_2, g_1^a)$. His goal is to find g_2^a (but he does not know s).

$\mathcal{B}$ sets $\text{pk} = (\mathbb{Z}_p^*, g_1, g_2)$ as the public key that $\mathcal{A}$ attacks in an IND-CPA game against Π_0.

G-queries: In the IND-CPA game, $\mathcal{A}$ asks for encryptions of messages $\Rightarrow$ $\mathcal{A}$ makes hash queries to G

- $\mathcal{B}$ simulates G by maintaining a list G_L of queries (Q, κ)
- i-th query Q_i: If Q_i in list, answer with (Q_i, κ_i); if not, pick randomly κ_i and add (Q_i, κ_i) to list

An IND-CPA secure PKE - generic construction

IND-CPA security: If the YC problem is hard then Π_0 is IND-CPA secure in the random oracle model (ROM).

Sketch of proof: Suppose $\mathcal{A}$ has non-negligible advantage against Π_0 in an IND-CPA game (can (t, ϵ) break it). We construct simulator $\mathcal{B}$ that breaks the YC problem with non-negligible probability.

Setup: $\mathcal{B}$ is given a YC instance $(\mathbb{Z}_p^*, g_1, g_2, g_1^a)$. His goal is to find g_2^a (but he does not know s).

$\mathcal{B}$ sets $\text{pk} = (\mathbb{Z}_p^*, g_1, g_2)$ as the public key that $\mathcal{A}$ attacks in an IND-CPA game against Π_0.

G-queries: In the IND-CPA game, $\mathcal{A}$ asks for encryptions of messages $\Rightarrow$ $\mathcal{A}$ makes hash queries to G

- $\mathcal{B}$ simulates G by maintaining a list G_L of queries (Q, κ)
- i-th query Q_i: If Q_i in list, answer with (Q_i, κ_i); if not, pick randomly κ_i and add (Q_i, κ_i) to list
- Crucial observation: If G is set as random oracle, κ is random and independent of Q, and **unknown** to $\mathcal{A}$, if it does not query the random oracle

An IND-CPA secure PKE - generic construction

IND-CPA security: If the YC problem is hard then Π_0 is IND-CPA secure in the random oracle model (ROM).

Sketch of proof: Suppose $\mathcal{A}$ has non-negligible advantage against Π_0 in an IND-CPA game (can (t, ϵ) break it). We construct simulator $\mathcal{B}$ that breaks the YC problem with non-negligible probability.

Setup: $\mathcal{B}$ is given a YC instance $(\mathbb{Z}_p^*, g_1, g_2, g_1^a)$. His goal is to find g_2^a (but he does not know s).

$\mathcal{B}$ sets $\mathsf{pk} = (\mathbb{Z}_p^*, g_1, g_2)$ as the public key that $\mathcal{A}$ attacks in an IND-CPA game against Π_0.

G-queries: In the IND-CPA game, $\mathcal{A}$ asks for encryptions of messages $\Rightarrow$ $\mathcal{A}$ makes hash queries to G

- $\mathcal{B}$ simulates G by maintaining a list G_L of queries (Q, κ)
- i-th query Q_i: If Q_i in list, answer with (Q_i, κ_i); if not, pick randomly κ_i and add (Q_i, κ_i) to list
- Crucial observation: If G is set as random oracle, κ is random and independent of Q, and **unknown** to $\mathcal{A}$, if it does not query the random oracle
- **Idea of proof**: Adversary has **NO** advantage in guessing the encrypted message without making a particular query Q^* - challenge query

Sketch of proof, contd.:

Challenge: $(M_0, M_1) \leftarrow \mathcal{A}(\mathrm{pk})$

$$\mathcal{B} : b \xleftarrow{\$} \{0,1\}, \quad \kappa^* \xleftarrow{\$} \{0,1\}^{\ell}, \quad C^* = (g_1^a, \kappa^* \oplus M_b)$$

- The challenge ciphertext C^* can be seen as encryption of M_b iff $\kappa^* = G(g_2^a)$ (see def. of Π_0)
- If adversary $\mathcal{A}$ has not queried $Q^* = g_2^a$, then $\kappa^* \oplus M_b$ is OTP encryption with unknown key κ^*

Sketch of proof, contd.:

Challenge: $(M_0, M_1) \leftarrow \mathcal{A}(\mathrm{pk})$

$$\mathcal{B} : b \xleftarrow{\$} \{0,1\}, \quad \kappa^* \xleftarrow{\$} \{0,1\}^\ell, \quad C^* = (g_1^a, \kappa^* \oplus M_b)$$

- The challenge ciphertext C^* can be seen as encryption of M_b iff $\kappa^* = G(g_2^a)$ (see def. of Π_0)
- If adversary $\mathcal{A}$ has not queried $Q^* = g_2^a$, then $\kappa^* \oplus M_b$ is OTP encryption with unknown key κ^*
- $\Rightarrow \mathcal{A}$ has no advantage in guessing M_b

An IND-CPA secure PKE - generic construction

Sketch of proof, contd.:

Challenge: $(M_0, M_1) \leftarrow \mathcal{A}(\mathrm{pk})$

$$\mathcal{B} : b \xleftarrow{\$} \{0,1\}, \quad \kappa^* \xleftarrow{\$} \{0,1\}^\ell, \quad C^* = (g_1^a, \kappa^* \oplus M_b)$$

- The challenge ciphertext C^* can be seen as encryption of M_b iff $\kappa^* = G(g_2^a)$ (see def. of Π_0)
- If adversary $\mathcal{A}$ has not queried $Q^* = g_2^a$, then $\kappa^* \oplus M_b$ is OTP encryption with unknown key κ^*
- $\Rightarrow \mathcal{A}$ has no advantage in guessing M_b
- $\Rightarrow \mathcal{A}$ must have queried the challenge query $Q^* = g_2^a$

Sketch of proof, contd.:

Challenge: $(M_0, M_1) \leftarrow \mathcal{A}(\mathsf{pk})$

$$\mathcal{B} : b \xleftarrow{\$} \{0,1\}, \quad \kappa^* \xleftarrow{\$} \{0,1\}^{\ell}, \quad C^* = (g_1^a, \kappa^* \oplus M_b)$$

- The challenge ciphertext C^* can be seen as encryption of M_b iff $\kappa^* = G(g_2^a)$ (see def. of Π_0)
- If adversary $\mathcal{A}$ has not queried $Q^* = g_2^a$, then $\kappa^* \oplus M_b$ is OTP encryption with unknown key κ^*
- $\Rightarrow$ $\mathcal{A}$ has no advantage in guessing M_b
- $\Rightarrow$ $\mathcal{A}$ must have queried the challenge query $Q^* = g_2^a$
- $\Rightarrow$ (Q^*, κ^*) must be in the list G_L

Sketch of proof, contd.:

Challenge: $(M_0, M_1) \leftarrow \mathcal{A}(\mathrm{pk})$

$$\mathcal{B} : b \xleftarrow{\$} \{0,1\}, \quad \kappa^* \xleftarrow{\$} \{0,1\}^{\ell}, \quad C^* = (g_1^a, \kappa^* \oplus M_b)$$

- The challenge ciphertext C^* can be seen as encryption of M_b iff $\kappa^* = G(g_2^a)$ (see def. of Π_0)
- If adversary $\mathcal{A}$ has not queried $Q^* = g_2^a$, then $\kappa^* \oplus M_b$ is OTP encryption with unknown key κ^*
- $\Rightarrow \mathcal{A}$ has no advantage in guessing M_b
- $\Rightarrow \mathcal{A}$ must have queried the challenge query $Q^* = g_2^a$
- $\Rightarrow (Q^*, \kappa^*)$ must be in the list G_L

Guess: $\mathcal{A}$ outputs a guess in the IND-CPA game

An IND-CPA secure PKE - generic construction

Sketch of proof, contd.:

Challenge: $(M_0, M_1) \leftarrow \mathcal{A}(\mathsf{pk})$

$$\mathcal{B} : b \xleftarrow{\$} \{0,1\}, \quad \kappa^* \xleftarrow{\$} \{0,1\}^\ell, \quad C^* = (g_1^a, \kappa^* \oplus M_b)$$

- The challenge ciphertext C^* can be seen as encryption of M_b iff $\kappa^* = G(g_2^a)$ (see def. of Π_0)
- If adversary $\mathcal{A}$ has not queried $Q^* = g_2^a$, then $\kappa^* \oplus M_b$ is OTP encryption with unknown key κ^*
- $\Rightarrow \mathcal{A}$ has no advantage in guessing M_b
- $\Rightarrow \mathcal{A}$ must have queried the challenge query $Q^* = g_2^a$
- $\Rightarrow$ (Q^*, κ^*) must be in the list G_L

Guess: $\mathcal{A}$ outputs a guess in the IND-CPA game

Output: $\mathcal{B}$ randomly selects an element (Q_{i^*}, κ_{i^*}) from G_L and outputs Q_{i^*}

An IND-CPA secure PKE - generic construction

Sketch of proof, contd.:

Challenge: $(M_0, M_1) \leftarrow \mathcal{A}(\mathrm{pk})$

$$\mathcal{B} : b \xleftarrow{\$} \{0,1\}, \quad \kappa^* \xleftarrow{\$} \{0,1\}^\ell, \quad C^* = (g_1^a, \kappa^* \oplus M_b)$$

- The challenge ciphertext C^* can be seen as encryption of M_b iff $\kappa^* = G(g_2^a)$ (see def. of Π_0)
- If adversary $\mathcal{A}$ has not queried $Q^* = g_2^a$, then $\kappa^* \oplus M_b$ is OTP encryption with unknown key κ^*
- $\Rightarrow$ $\mathcal{A}$ has no advantage in guessing M_b
- $\Rightarrow$ $\mathcal{A}$ must have queried the challenge query $Q^* = g_2^a$
- $\Rightarrow$ (Q^*, κ^*) must be in the list G_L

Guess: $\mathcal{A}$ outputs a guess in the IND-CPA game

Output: $\mathcal{B}$ randomly selects an element (Q_{i^*}, κ_{i^*}) from G_L and outputs Q_{i^*}

- Advantage of breaking YC: ϵ/q_G, q_G - number of queries to G and ϵ- advantage of $\mathcal{A}$ against Π_0
- Cost: $t + T_s$, T_s - cost of simulation
- $\Rightarrow$ The adversary $\mathcal{B}$ $(t + T_s, \epsilon/q_G)$ solves YC

From IND-CPA to IND-CCA PKE - generic construction

Fujisaki-Okamoto first transform: Let $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be IND-CPA secure PKE.

We define the transformed $\Pi' = (\mathsf{KGen}', \mathsf{Enc}^H, \mathsf{Dec}^H)$ as:

- $\mathsf{KGen}'(1^k)$ just runs $\mathsf{KGen}(1^k)$
- We need $H : \{0,1\}^* \to \{0,1\}^\ell$
- Enc^H: Choose $R \xleftarrow{\$} \{0,1\}^{k_0}$ and compute $C \leftarrow \mathsf{Enc}^H(M, R) = \mathsf{Enc}(M||R, H(M||R))$
- Dec^H: Compute $M'||R' = \mathsf{Dec}(C)$ and output M' if $\mathsf{Enc}^H(M', R') = C$, and $\perp$ otherwise

From IND-CPA to IND-CCA PKE - generic construction

Fujisaki-Okamoto first transform: Let $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be IND-CPA secure PKE.
We define the transformed $\Pi' = (\mathsf{KGen}', \mathsf{Enc}^H, \mathsf{Dec}^H)$ as:

- $\mathsf{KGen}'(1^k)$ just runs $\mathsf{KGen}(1^k)$
- We need $H : \{0,1\}^* \to \{0,1\}^\ell$
- Enc^H: Choose $R \xleftarrow{\$} \{0,1\}^{k_0}$ and compute $C \leftarrow \mathsf{Enc}^H(M, R) = \mathsf{Enc}(M||R, H(M||R))$
- Dec^H: Compute $M'||R' = \mathsf{Dec}(C)$ and output M' if $\mathsf{Enc}^H(M', R') = C$, and $\perp$ otherwise

IND-CC2 security: If $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CPA secure PKE ($+$ another standard property) then $\Pi' = (\mathsf{KGen}', \mathsf{Enc}^H, \mathsf{Dec}^H)$ is IND-CCA2 secure in the random oracle model.

From IND-CPA to IND-CCA PKE - generic construction

Fujisaki-Okamoto first transform: Let $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be IND-CPA secure PKE.
We define the transformed $\Pi' = (\mathsf{KGen}', \mathsf{Enc}^H, \mathsf{Dec}^H)$ as:

- $\mathsf{KGen}'(1^k)$ just runs $\mathsf{KGen}(1^k)$
- We need $H : \{0,1\}^* \to \{0,1\}^\ell$
- Enc^H: Choose $R \xleftarrow{\$} \{0,1\}^{k_0}$ and compute $C \leftarrow \mathsf{Enc}^H(M, R) = \mathsf{Enc}(M||R, H(M||R))$
- Dec^H: Compute $M'||R' = \mathsf{Dec}(C)$ and output M' if $\mathsf{Enc}^H(M', R') = C$, and $\perp$ otherwise

IND-CC2 security: If $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CPA secure PKE ($+$ another standard property) then $\Pi' = (\mathsf{KGen}', \mathsf{Enc}^H, \mathsf{Dec}^H)$ is IND-CCA2 secure in the random oracle model.

Some remarks:

- reduction loss of q_H- number of queries to oracle H
- Needs IND-CPA of starting scheme - quite strong to begin with

From IND-CPA to IND-CCA PKE - generic construction

Fujisaki-Okamoto first transform: Let $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ be IND-CPA secure PKE.
We define the transformed $\Pi' = (\mathsf{KGen}', \mathsf{Enc}^H, \mathsf{Dec}^H)$ as:

- $\mathsf{KGen}'(1^k)$ just runs $\mathsf{KGen}(1^k)$
- We need $H : \{0,1\}^* \to \{0,1\}^\ell$
- Enc^H: Choose $R \xleftarrow{\$} \{0,1\}^{k_0}$ and compute $C \leftarrow \mathsf{Enc}^H(M, R) = \mathsf{Enc}(M||R, H(M||R))$
- Dec^H: Compute $M'||R' = \mathsf{Dec}(C)$ and output M' if $\mathsf{Enc}^H(M', R') = C$, and $\perp$ otherwise

IND-CC2 security: If $\Pi = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CPA secure PKE ($+$ another standard property) then $\Pi' = (\mathsf{KGen}', \mathsf{Enc}^H, \mathsf{Dec}^H)$ is IND-CCA2 secure in the random oracle model.

Some remarks:

- reduction loss of q_H- number of queries to oracle H
- Needs IND-CPA of starting scheme - quite strong to begin with
- We need conversions from weaker security guarantees

In practice ...

- Public key encryption typically not used in practice
- Typically: transport symmetric key using public key crypto, then encrypt traffic using symmetric crypto

In practice ...

- Public key encryption typically not used in practice
- Typically: transport symmetric key using public key crypto, then encrypt traffic using symmetric crypto
- But ... Recall the problems of small messages in textbook RSA

In practice ...

- Public key encryption typically not used in practice
- Typically: transport symmetric key using public key crypto, then encrypt traffic using symmetric crypto
- But ... Recall the problems of small messages in textbook RSA
- Solution (Shoup): Hash the message after decryption and then use as a symmetric key

In practice ...

- Public key encryption typically not used in practice
- Typically: transport symmetric key using public key crypto, then encrypt traffic using symmetric crypto
- But ... Recall the problems of small messages in textbook RSA
- Solution (Shoup): Hash the message after decryption and then use as a symmetric key

Hybrid scheme:

Key Encapsulation Mechanism (KEM)

$+$

Data Encapsulation Mechanism (DEM)

- KEM - definition and security similar to PKE
- DEM - basically symmetric key encryption (definition and security)

Key Encapsulation Mechanism (KEM) – definition

Given security parameter $\lambda \in \mathbb{N}$ and two finite sets $\mathcal{M}, \mathcal{R} \subseteq \{0,1\}^*$, a Key Encapsulation Mechanism (KEM) $\Pi = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ consists of three algorithms:

- **Key-generation algorithm** (probabilistic): $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
- **Encapsulation algorithm** (probabilistic): Takes random $r \in \mathcal{R}$ and outputs $(K, C) \leftarrow \mathsf{Encaps}(\mathsf{pk}, M, r)$. C is said to be the encapsulation of key $K \in \mathcal{K}$.
- **Decapsulation algorithm** (deterministic): Takes as input a secret key sk and encapsulation C, and outputs either a key $K' = \mathsf{Decaps}(\mathsf{sk}, C) \in \mathcal{K}$ or $\perp \notin \mathcal{K}$ to indicate an invalid encapsulation.

Key Encapsulation Mechanism (KEM) – definition

Given security parameter $\lambda \in \mathbb{N}$ and two finite sets $\mathcal{M}, \mathcal{R} \subseteq \{0,1\}^*$, a Key Encapsulation Mechanism (KEM) $\Pi = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ consists of three algorithms:

- **Key-generation algorithm** (probabilistic): $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
- **Encapsulation algorithm** (probabilistic): Takes random $r \in \mathcal{R}$ and outputs $(K, C) \leftarrow \mathsf{Encaps}(\mathsf{pk}, M, r)$. C is said to be the encapsulation of key $K \in \mathcal{K}$.
- **Decapsulation algorithm** (deterministic): Takes as input a secret key sk and encapsulation C, and outputs either a key $K' = \mathsf{Decaps}(\mathsf{sk}, C) \in \mathcal{K}$ or $\perp \notin \mathcal{K}$ to indicate an invalid encapsulation.

Correctness: For all $K \in \mathcal{K}$

$$Pr[\mathsf{Decaps}(\mathsf{sk}, C) = K : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda), C \leftarrow \mathsf{Encaps}(\mathsf{pk}, r)] \geqslant 1 - \delta$$

for a decryption error δ.

Security of Key Encapsulation Mechanism (KEM)

A KEM scheme *KEM* is called IND-CCA2-secure if any PPT algorithm $\mathcal{A}$ has only negligible advantage

$$Adv = \mathbf{Pr}\left(\mathrm{Exp}^{\text{ind-cca}}_{KEM(1^k)}(\mathcal{A}) = 1\right) - 1/2 = negl(k)\,.$$

in the following $\mathrm{Exp}^{\textbf{ind-cca}}_{KEM(1^k)}(\mathcal{A})$ **game (experiment)**:

Challenger		**Adversary**
$(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KGen}()$	$\xrightarrow{\mathrm{pk}}$	
	$\xleftarrow{C_i}$	
	$\xrightarrow{\mathrm{Decaps}(\mathrm{sk}, C_i)}$	
$(K_0, C) \leftarrow \mathrm{Encaps}(\mathrm{pk})$		
$K_1 \xleftarrow{\$} \mathcal{K}$		
$b \xleftarrow{\$} \{0,1\}$	$\xrightarrow{C, K_b}$	
	$\xleftarrow{C_i}$	
	$\xrightarrow{\mathrm{Decaps}(\mathrm{sk}, C_i)}$	
	$\xleftarrow{b'}$	b'
Return 1 iff $b = b'$ otherwise 0.		

Fujisaki Okamoto second transform (KEM version)

Fujisaki and Okamoto proposed another transform (in this course we call it second)

- requires only very weak notion of OW-CPA security of PKE

A probabilistic encryption scheme $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be **one-way** (OW-CPA) if the probability that a polynomial time attacker $\mathcal{A}$ can invert a ciphertext $C = \mathsf{Enc}(M; \mathsf{pk})$ obtained by encrypting a random message M, is negligible

Fujisaki Okamoto second transform (KEM version)

Fujisaki and Okamoto proposed another transform (in this course we call it second)

- requires only very weak notion of OW-CPA security of PKE

A probabilistic encryption scheme $(\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ is said to be **one-way** (OW-CPA) if the probability that a polynomial time attacker $\mathcal{A}$ can invert a ciphertext $C = \mathsf{Enc}(M; \mathsf{pk})$ obtained by encrypting a random message M, is negligible

- transform originally proposed for IND-CCA2 security of PKE
- here we look at KEM version (Dent 2003) from probabilistic OW-CPA PKE
  - version exists from deterministic PKE, and different security properties of the PKE

Fujisaki Okamoto second transform (KEM version)

Fujisaki and Okamoto proposed another transform (in this course we call it second)

- requires only very weak notion of OW-CPA security of PKE

A probabilistic encryption scheme (KGen, Enc, Dec) is said to be **one-way** (OW-CPA) if the probability that a polynomial time attacker $\mathcal{A}$ can invert a ciphertext $C = \mathsf{Enc}(M; \mathsf{pk})$ obtained by encrypting a random message M, is negligible

- transform originally proposed for IND-CCA2 security of PKE
- here we look at KEM version (Dent 2003) from probabilistic OW-CPA PKE
  - version exists from deterministic PKE, and different security properties of the PKE
- We need $H : \{0,1\}^* \rightarrow \{0,1\}^k$ and key derivation function KDF
  - both modelled as random oracles
  - in practice caution about their instantiations

Fujisaki Okamoto second transform (KEM version)

FO-KEM: Let $(\mathsf{KGen}_E, \mathsf{Enc}, \mathsf{Dec})$ be OW-CPA secure PKE.

We define $FO_{KEM} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ as:

- $\mathsf{KGen}(1^k)$ just runs $\mathsf{KGen}_E(1^k)$
- Encaps:
  - Choose $X \xleftarrow{\$} \mathcal{M}$, set $R = H(X)$ and compute $C \leftarrow \mathsf{Enc}(X, R)$ (make deterministic)
  - Set $K = KDF(X)$ and output (K, C)
- Decaps:
  - Set $X \leftarrow \mathsf{Dec}(C)$. If $X = \perp$, output $\perp$ and halt.
  - Set $R = H(X)$
  - Check $C \stackrel{?}{=} \mathsf{Enc}(X, R)$. If not, output $\perp$ and halt. (re-encryption)
  - Set $K = KDF(X)$ and output (K, C)

Fujisaki Okamoto second transform (KEM version)

FO-KEM: Let $(\mathsf{KGen}_E, \mathsf{Enc}, \mathsf{Dec})$ be OW-CPA secure PKE.

We define $FO_{KEM} = (\mathsf{KGen}, \mathsf{Encaps}, \mathsf{Decaps})$ as:

- $\mathsf{KGen}(1^k)$ just runs $\mathsf{KGen}_E(1^k)$
- Encaps:
  - Choose $X \xleftarrow{\$} \mathcal{M}$, set $R = H(X)$ and compute $C \leftarrow \mathsf{Enc}(X, R)$ (make deterministic)
  - Set $K = KDF(X)$ and output (K, C)
- Decaps:
  - Set $X \leftarrow \mathsf{Dec}(C)$. If $X = \perp$, output $\perp$ and halt.
  - Set $R = H(X)$
  - Check $C \stackrel{?}{=} \mathsf{Enc}(X, R)$. If not, output $\perp$ and halt. (re-encryption)
  - Set $K = KDF(X)$ and output (K, C)

IND-CCA2 security:

If $(\mathsf{KGen}_E, \mathsf{Enc}, \mathsf{Dec})$ is OW-CPA secure PKE, then **FO-KEM is IND-CCA2 secure** in the ROM.

Other transforms and standards

- Other generic transforms exist: REACT, GEM [OP01]
- Recently, a unified framework [HHK17] puts all of them under FO-transforms
- FO transforms very relevant for modern cryptosystems (post-quantum cryptosystems)
- to be standardized via Kyber (now ML-KEM, draft standard out in '23), but other schemes expected in the near future

Other transforms and standards

- Other generic transforms exist: REACT, GEM [OP01]
- Recently, a unified framework [HHK17] puts all of them under FO-transforms
- FO transforms very relevant for modern cryptosystems (post-quantum cryptosystems)
- to be standardized via Kyber (now ML-KEM, draft standard out in '23), but other schemes expected in the near future

Other existing standards today:

- RSA-OAEP (NIST.SP.800-56Br2)
  - Bellare and Rogaway, 1994
  - very complex, initial proof wrong
  - provably secure under the RSA assumption (It is hard to find x, given $y = x^e \pmod{N}$, e and N.)

Other transforms and standards

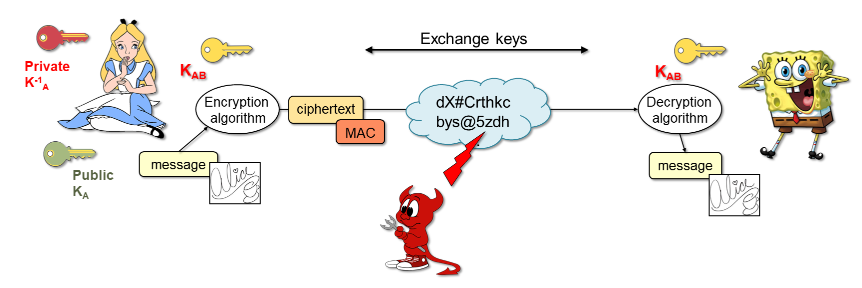
- Other generic transforms exist: REACT, GEM [OP01]
- Recently, a unified framework [HHK17] puts all of them under FO-transforms
- FO transforms very relevant for modern cryptosystems (post-quantum cryptosystems)
- to be standardized via Kyber (now ML-KEM, draft standard out in '23), but other schemes expected in the near future

Other existing standards today:

- RSA-OAEP (NIST.SP.800-56Br2)
  - Bellare and Rogaway, 1994
  - very complex, initial proof wrong
  - provably secure under the RSA assumption (It is hard to find x, given $y = x^e \pmod{N}$, e and N.)
- RSA-KEM (ISO/IEC18033-2)
  - Shoup
  - provably secure under the RSA assumption

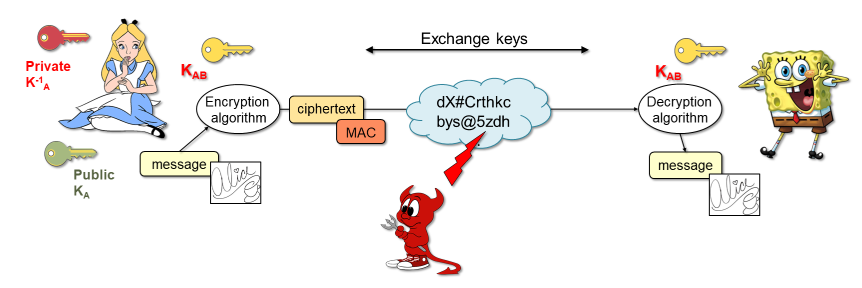
Digital signatures

In our everyday scenario



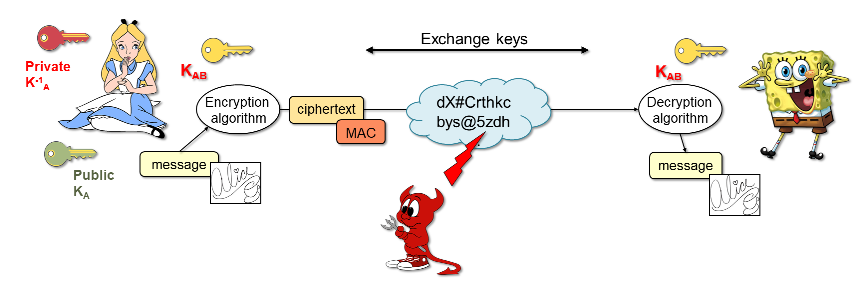
- Alice signs a message using her private key
  - for example in an authenticated key exchange

In our everyday scenario



- Alice signs a message using her private key
  - for example in an authenticated key exchange
- Bob verifies the signature using Alice's public key and the message

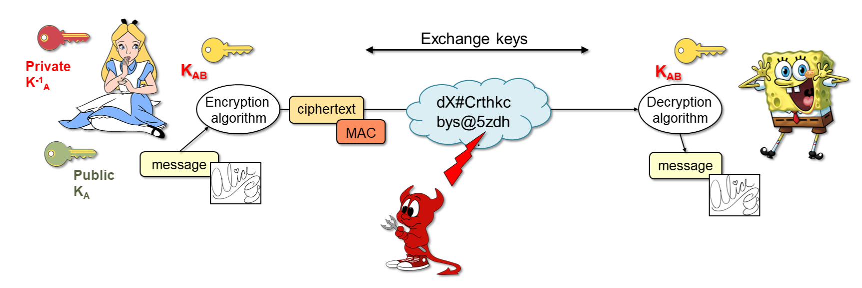
In our everyday scenario



- Alice signs a message using her private key
  - for example in an authenticated key exchange
- Bob verifies the signature using Alice's public key and the message

- What does Eve want to do/achieve?

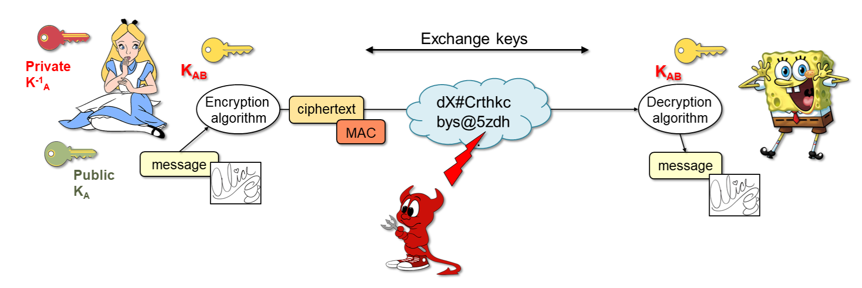
In our everyday scenario



- Alice signs a message using her private key
    - for example in an authenticated key exchange
- Bob verifies the signature using Alice's public key and the message

- What does Eve want to do/achieve?
    - **Forge a signature!**

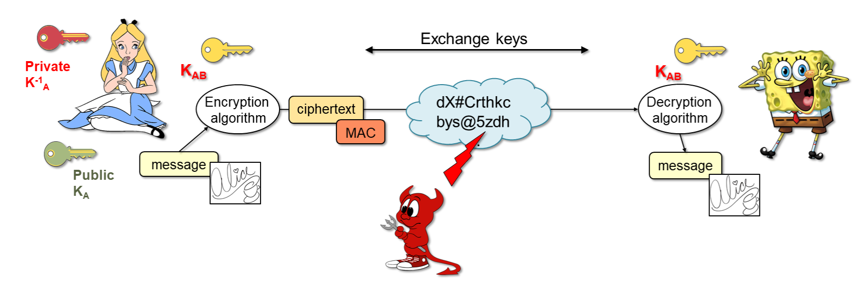
In our everyday scenario



- Alice signs a message using her private key
  - for example in an authenticated key exchange
- Bob verifies the signature using Alice's public key and the message

- What does Eve want to do/achieve?
  - **Forge a signature!**
  - **Ultimate goal:** Recover **private key**! Then forge signature for **ALL** messages!

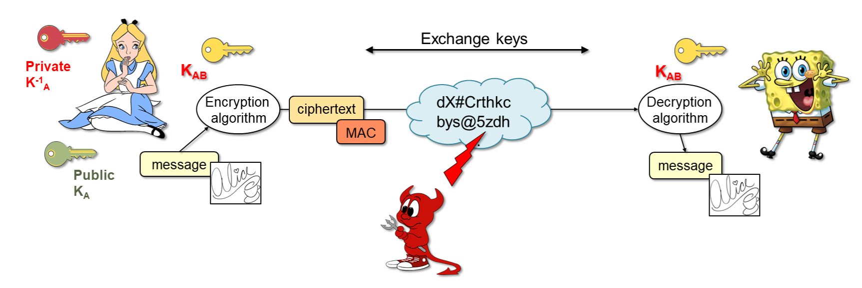
In our everyday scenario



- Alice signs a message using her private key
  - for example in an authenticated key exchange
- Bob verifies the signature using Alice's public key and the message

- What does Eve want to do/achieve?
  - **Forge a signature!**
  - **Ultimate goal:** Recover **private key**! Then forge signature for **ALL** messages!
  - **Excellent:** Forge signature for **ANY** message!

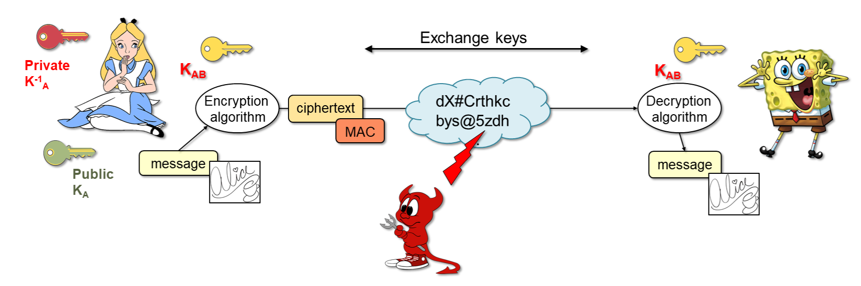
In our everyday scenario



- Alice signs a message using her private key
  - for example in an authenticated key exchange
- Bob verifies the signature using Alice's public key and the message

- What does Eve want to do/achieve?
  - **Forge a signature!**
  - **Ultimate goal:** Recover **private key**! Then forge signature for **ALL** messages!
  - **Excellent:** Forge signature for **ANY** message!
  - **Also good:** Forge signature for **SOME** message she chooses!

In our everyday scenario



- Alice signs a message using her private key
  - for example in an authenticated key exchange
- Bob verifies the signature using Alice's public key and the message

- What does Eve want to do/achieve?
  - **Forge a signature!**
  - **Ultimate goal:** Recover **private key**! Then forge signature for **ALL** messages!
  - **Excellent:** Forge signature for **ANY** message!
  - **Also good:** Forge signature for **SOME** message she chooses!
  - **Satisfactory:** Forge signature for **ONE** gibberish message!

Digital Signatures (DSs) – definition

Given security parameter $\lambda \in \mathbb{N}$ and message space $\mathcal{M} \subseteq \{0,1\}^*$, a digital signature scheme $DSs = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ consists of three PPT algorithms:

- **Key-generation algorithm** (probabilistic): $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
- **Signing algorithm** (probabilistic): Takes $M \in \mathcal{M}$ and secret key sk and outputs a signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, M)$
- **Verification algorithm** (deterministic): Takes as input a public key pk, message M, and signature σ and outputs $Accept \leftarrow \mathsf{Vf}(\mathsf{pk}, M, \sigma)$ if σ is a valid signature of M under the public key pk or $\perp$ otherwise.

Digital Signatures (DSs) – definition

Given security parameter $\lambda \in \mathbb{N}$ and message space $\mathcal{M} \subseteq \{0,1\}^*$, a digital signature scheme $DSs = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ consists of three PPT algorithms:

- **Key-generation algorithm** (probabilistic): $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
- **Signing algorithm** (probabilistic): Takes $M \in \mathcal{M}$ and secret key sk and outputs a signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, M)$
- **Verification algorithm** (deterministic): Takes as input a public key pk, message M, and signature σ and outputs $Accept \leftarrow \mathsf{Vf}(\mathsf{pk}, M, \sigma)$ if σ is a valid signature of M under the public key pk or $\perp$ otherwise.

Correctness: For all $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$ and all $M \in \mathcal{M}$:

$$\mathsf{Vf}(\mathsf{pk}, M, \mathsf{Sign}(\mathsf{sk}, M)) = Accept$$

Digital Signatures (DSs) – definition

Given security parameter $\lambda \in \mathbb{N}$ and message space $\mathcal{M} \subseteq \{0,1\}^*$, a digital signature scheme $DSs = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ consists of three PPT algorithms:

- **Key-generation algorithm** (probabilistic): $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
- **Signing algorithm** (probabilistic): Takes $M \in \mathcal{M}$ and secret key sk and outputs a signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, M)$
- **Verification algorithm** (deterministic): Takes as input a public key pk, message M, and signature σ and outputs $Accept \leftarrow \mathsf{Vf}(\mathsf{pk}, M, \sigma)$ if σ is a valid signature of M under the public key pk or $\perp$ otherwise.

Correctness: For all $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$ and all $M \in \mathcal{M}$:

$$\mathsf{Vf}(\mathsf{pk}, M, \mathsf{Sign}(\mathsf{sk}, M)) = Accept$$

- **Passive attacker** - **observes** none/some/many signatures of messages

Digital Signatures (DSs) – definition

Given security parameter $\lambda \in \mathbb{N}$ and message space $\mathcal{M} \subseteq \{0,1\}^*$, a digital signature scheme $DSs = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ consists of three PPT algorithms:

- **Key-generation algorithm** (probabilistic): $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
- **Signing algorithm** (probabilistic): Takes $M \in \mathcal{M}$ and secret key sk and outputs a signature $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, M)$
- **Verification algorithm** (deterministic): Takes as input a public key pk, message M, and signature σ and outputs $Accept \leftarrow \mathsf{Vf}(\mathsf{pk}, M, \sigma)$ if σ is a valid signature of M under the public key pk or $\perp$ otherwise.

Correctness: For all $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$ and all $M \in \mathcal{M}$:

$$\mathsf{Vf}(\mathsf{pk}, M, \mathsf{Sign}(\mathsf{sk}, M)) = Accept$$

- **Passive attacker** - **observes** none/some/many signatures of messages
- **Active attacker** - **can craft messages** to send to **signing oracle** to be signed!

Security of Digital Signatures (DSs)

Standard security: Existential unforgeability under adaptive chosen message attacks (EUF-CMA)

A Digital Signature scheme *Dss* is called EUF-CMA-secure if any PPT algorithm $\mathcal{A}$ has only negligible success probability

$$\mathrm{Succ}^{\text{euf-cma}}_{\mathsf{DSs}(1^k)}(\mathcal{A}) = Pr\left[\mathsf{Exp}^{\text{euf-cma}}_{\mathsf{DSs}(1^k)}(\mathcal{A}) = Accept\right].$$

in the following $\mathsf{Exp}^{\text{euf-cma}}_{\mathsf{DSs}(1^k)}(\mathcal{A})$ **game (experiment)**:

Challenger		**Adversary**
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}()$	$\xrightarrow{\mathsf{pk}}$	
	$\xleftarrow{M_i}$	M_i for number of i-s
	$\xrightarrow{\mathsf{Sign}(\mathsf{sk}, M_i)}$	
	$\xleftarrow{(M^*, \sigma^*)}$	M^*, σ^*

Return 1 iff $\mathsf{Vf}(\mathsf{pk}, M, \sigma) = Accept$
otherwise 0.

An example walkthrough - RSA signatures

Textbook RSA signature (directly from RSA encryption algorithm):

Textbook RSA:

KeyGen:

1. Choose two primes p, q s.t. $|p| \approx |q|$
2. Compute $N = pq$ and $\phi(N) = (p-1)(q-1)$
3. Choose a random $e < \phi(N)$, s.t. $\gcd(e, \phi(N)) = 1$
4. Compute d such that $ed = 1 \pmod{\phi(N)}$
5. Output public key $\mathsf{pk} = (N, e)$ and private key $\mathsf{sk} = d$

Sign:

Given message M, compute signature by $\sigma \leftarrow M^d \pmod{N}$

Verify:

To verify the message - signature pair (M, σ) compute $M' \leftarrow \sigma^e \pmod{N}$

If $M' = M$ output *Accept*

Trivial existential forgery attack:

- Eve knows only the public key (N, e)

An example walkthrough - forgeries on RSA signatures

Trivial existential forgery attack:

- Eve knows only the public key (N, e)
- Eve chooses random $\sigma \in \mathbb{Z}_N$, and calculates $M = \sigma^e$

An example walkthrough - forgeries on RSA signatures

Trivial existential forgery attack:

- Eve knows only the public key (N, e)
- Eve chooses random $\sigma \in \mathbb{Z}_N$, and calculates $M = \sigma^e$
  - (M, σ) is a **valid signature pair**!

An example walkthrough - forgeries on RSA signatures

Trivial existential forgery attack:

- Eve knows only the public key (N, e)
- Eve chooses random $\sigma \in \mathbb{Z}_N$, and calculates $M = \sigma^e$
  - (M, σ) is a **valid signature pair**!
- **Does not even require access to signing oracle!** (Key Only Attack (KOA))

Trivial existential forgery attack:

- Eve knows only the public key (N, e)
- Eve chooses random $\sigma \in \mathbb{Z}_N$, and calculates $M = \sigma^e$
  - (M, σ) is a **valid signature pair**!
- **Does not even require access to signing oracle!** (Key Only Attack (KOA))
- Not specific only to textbook RSA, but to any scheme whose verification algorithm can efficiently **compute** the message M from the signature σ

An example walkthrough - forgeries on RSA signatures

Trivial existential forgery attack:

- Eve knows only the public key (N, e)
- Eve chooses random $\sigma \in \mathbb{Z}_N$, and calculates $M = \sigma^e$
  - (M, σ) is a **valid signature pair**!
- **Does not even require access to signing oracle!** (Key Only Attack (KOA))
- Not specific only to textbook RSA, but to any scheme whose verification algorithm can efficiently **compute** the message M from the signature σ

Chosen message universal forgery attack:

- To forge a message M that is composite i.e. $M = M_1 M_2 \pmod{N}$:

An example walkthrough - forgeries on RSA signatures

Trivial existential forgery attack:

- Eve knows only the public key (N, e)
- Eve chooses random $\sigma \in \mathbb{Z}_N$, and calculates $M = \sigma^e$
    - (M, σ) is a **valid signature pair**!
- **Does not even require access to signing oracle!** (Key Only Attack (KOA))
- Not specific only to textbook RSA, but to any scheme whose verification algorithm can efficiently **compute** the message M from the signature σ

Chosen message universal forgery attack:

- To forge a message M that is composite i.e. $M = M_1 M_2 \pmod N$:
- Eve asks for the signatures σ_1 and σ_2 of M_1 ad M_2

An example walkthrough - forgeries on RSA signatures

Trivial existential forgery attack:

- Eve knows only the public key (N, e)
- Eve chooses random $\sigma \in \mathbb{Z}_N$, and calculates $M = \sigma^e$
  - (M, σ) is a **valid signature pair**!
- **Does not even require access to signing oracle!** (Key Only Attack (KOA))
- Not specific only to textbook RSA, but to any scheme whose verification algorithm can efficiently **compute** the message M from the signature σ

Chosen message universal forgery attack:

- To forge a message M that is composite i.e. $M = M_1 M_2 \pmod N$:
- Eve asks for the signatures σ_1 and σ_2 of M_1 ad M_2
- $(M, \sigma_1 \sigma_2)$ is a **valid signature pair** because of multiplicativity!
- Not specific only to textbook RSA, but to any scheme that is multiplicative

$$\mathsf{Sign}(M_1) \cdot \mathsf{Sign}(M_2) = \mathsf{Sign}(M_1 \cdot M_2)$$

An example walkthrough - forgeries on RSA signatures

Solution?

- Make sure the two properties not satisfied

Solution?

- Make sure the two properties not satisfied
  - message M efficiently computable from the signature σ
  - multiplicativity

Solution?

- Make sure the two properties not satisfied
  - message M efficiently computable from the signature σ
  - multiplicativity
- Simple hashing should suffice, right?
  - at the time ($\sim$ 20 years ago) MD5 or SHA1
  - Take $\mu(M) = H(M)$ for H a hash function of length 128 or 160 bits

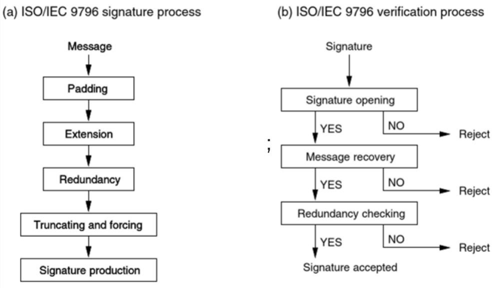
An example walkthrough - forgeries on RSA signatures

Solution?

- Make sure the two properties not satisfied
  - message M efficiently computable from the signature σ
  - multiplicativity
- Simple hashing should suffice, right?
  - at the time ($\sim$ 20 years ago) MD5 or SHA1
  - Take $\mu(M) = H(M)$ for H a hash function of length 128 or 160 bits
  - basis of the **ISO/IEC 9796 standard**, that employs a more complicated **padding scheme** $\mu(M) = \mathit{ComplicatedPadding}(H(M))$

Solution?

- Make sure the two properties not satisfied
  - message M efficiently computable from the signature σ
  - multiplicativity
- Simple hashing should suffice, right?
  - at the time ($\sim$ 20 years ago) MD5 or SHA1
  - Take $\mu(M) = H(M)$ for H a hash function of length 128 or 160 bits
  - basis of the **ISO/IEC 9796 standard**, that employs a more complicated **padding scheme** $\mu(M) = ComplicatedPadding(H(M))$
- As you might expect, there is an attack!
  - we look at a simplified version



An example walkthrough - forgeries on RSA signatures

Attack on ISO/IEC 9796:[Coron, Naccache, Stern, 1999] (extension of Desmedt-Odlyzko attack)

Setup:

- Given public key $\mathrm{pk} = (N, e)$ and function $\mu(M) = H(M)$ where $H(M)$ is short (128 or 160 bits)

An example walkthrough - forgeries on RSA signatures

Attack on ISO/IEC 9796:[Coron, Naccache, Stern, 1999] (extension of Desmedt-Odlyzko attack)

Setup:

- Given public key $\mathsf{pk} = (N, e)$ and function $\mu(M) = H(M)$ where $H(M)$ is short (128 or 160 bits)
- $\boxed{\text{A positive integer } b \text{ is } \ell\textbf{-smooth} \text{ if all its prime factors are smaller than } \ell.}$

An example walkthrough - forgeries on RSA signatures

Attack on ISO/IEC 9796:[Coron, Naccache, Stern, 1999] (extension of Desmedt-Odlyzko attack)

Setup:

- Given public key $\mathsf{pk} = (N, e)$ and function $\mu(M) = H(M)$ where $H(M)$ is short (128 or 160 bits)
- $\boxed{\text{A positive integer } b \text{ is } \ell\text{-}\textbf{smooth} \text{ if all its prime factors are smaller than } \ell.}$
  - Probability that SHA-1 digest is 2^{24}-smooth is 2^{-19}, so quite feasible to find smooth digests

An example walkthrough - forgeries on RSA signatures

Attack on ISO/IEC 9796:[Coron, Naccache, Stern, 1999] (extension of Desmedt-Odlyzko attack)

Setup:

- Given public key pk $= (N, e)$ and function $\mu(M) = H(M)$ where $H(M)$ is short (128 or 160 bits)
- A positive integer b is **ℓ-smooth** if all its prime factors are smaller than ℓ.
  - Probability that SHA-1 digest is 2^{24}-smooth is 2^{-19}, so quite feasible to find smooth digests
- Let $\{p_1, p_2, \ldots, p_t\}$ be the set of the first t primes
  - We will consider p_t-smooth numbers, which can be expressed as

$$b = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_t^{\alpha_t}$$

An example walkthrough - Attack on ISO/IEC 9796 (CNS99)

The attack:

- Find $t+1$ messages $M_1, M_2, \ldots, M_{t+1}$ such that all $\mu(M_1), \mu(M_2), \ldots, \mu(M_{t+1})$ are smooth

The attack:

- Find $t+1$ messages $M_1, M_2, \ldots, M_{t+1}$ such that all $\mu(M_1), \mu(M_2), \ldots, \mu(M_{t+1})$ are smooth
- They can all be expressed as:
-

$$
\begin{aligned}
&\mu(M_1) = p_1^{\alpha_1^{(1)}} p_2^{\alpha_2^{(1)}} \ldots p_t^{\alpha_t^{(1)}} \\
&\mu(M_2) = p_1^{\alpha_1^{(2)}} p_2^{\alpha_2^{(2)}} \ldots p_t^{\alpha_t^{(2)}} \\
&\ldots \\
&\mu(M_{t+1}) = p_1^{\alpha_1^{(t+1)}} p_2^{\alpha_2^{(t+1)}} \ldots p_t^{\alpha_t^{(t+1)}}
\end{aligned}
$$

The attack:

- Find $t+1$ messages $M_1, M_2, \ldots, M_{t+1}$ such that all $\mu(M_1), \mu(M_2), \ldots, \mu(M_{t+1})$ are smooth
- They can all be expressed as:
- Consider only the vectors of the exponents $\pmod{e}$

$$
\begin{aligned}
\mu(M_1) &= p_1^{\alpha_1^{(1)}} p_2^{\alpha_2^{(1)}} \ldots p_t^{\alpha_t^{(1)}} &\longrightarrow\quad v_1 &= \left(\alpha_1^{(1)} \pmod{e},\ \alpha_2^{(1)} \pmod{e}, \ldots,\ \alpha_t^{(1)} \pmod{e}\right) \\
\mu(M_2) &= p_1^{\alpha_1^{(2)}} p_2^{\alpha_2^{(2)}} \ldots p_t^{\alpha_t^{(2)}} &\longrightarrow\quad v_2 &= \left(\alpha_1^{(2)} \pmod{e},\ \alpha_2^{(2)} \pmod{e}, \ldots,\ \alpha_t^{(2)} \pmod{e}\right) \\
&\ldots \\
\mu(M_{t+1}) &= p_1^{\alpha_1^{(t+1)}} p_2^{\alpha_2^{(t+1)}} \ldots p_t^{\alpha_t^{(t+1)}} &\longrightarrow\quad v_{t+1} &= \left(\alpha_1^{(t+1)} \pmod{e},\ \alpha_2^{(t+1)} \pmod{e}, \ldots,\ \alpha_t^{(t+1)} \pmod{e}\right)
\end{aligned}
$$

The attack:

- Find $t+1$ messages $M_1, M_2, \ldots, M_{t+1}$ such that all $\mu(M_1), \mu(M_2), \ldots, \mu(M_{t+1})$ are smooth
- They can all be expressed as:
- Consider only the vectors of the exponents $\pmod e$

$$\mu(M_1) = p_1^{\alpha_1^{(1)}} p_2^{\alpha_2^{(1)}} \ldots p_t^{\alpha_t^{(1)}} \longrightarrow v_1 = \left(\alpha_1^{(1)} \pmod e,\ \alpha_2^{(1)} \pmod e, \ldots,\ \alpha_t^{(1)} \pmod e\right)$$

$$\mu(M_2) = p_1^{\alpha_1^{(2)}} p_2^{\alpha_2^{(2)}} \ldots p_t^{\alpha_t^{(2)}} \longrightarrow v_2 = \left(\alpha_1^{(2)} \pmod e,\ \alpha_2^{(2)} \pmod e, \ldots,\ \alpha_t^{(2)} \pmod e\right)$$

$$\ldots$$

$$\mu(M_{t+1}) = p_1^{\alpha_1^{(t+1)}} p_2^{\alpha_2^{(t+1)}} \ldots p_t^{\alpha_t^{(t+1)}} \longrightarrow v_{t+1} = \left(\alpha_1^{(t+1)} \pmod e,\ \alpha_2^{(t+1)} \pmod e, \ldots,\ \alpha_t^{(t+1)} \pmod e\right)$$

Crucial observation:

- We have $t+1$ vectors in a space of dimension t

An example walkthrough - Attack on ISO/IEC 9796 (CNS99)

The attack:

- Find $t+1$ messages $M_1, M_2, \dots, M_{t+1}$ such that all $\mu(M_1), \mu(M_2), \dots, \mu(M_{t+1})$ are smooth
- They can all be expressed as:
- Consider only the vectors of the exponents $\pmod e$

$$
\begin{aligned}
\mu(M_1) &= p_1^{\alpha_1^{(1)}} p_2^{\alpha_2^{(1)}} \dots p_t^{\alpha_t^{(1)}} && \longrightarrow && v_1 = \left(\alpha_1^{(1)} \pmod e,\ \alpha_2^{(1)} \pmod e, \dots,\ \alpha_t^{(1)} \pmod e\right) \\
\mu(M_2) &= p_1^{\alpha_1^{(2)}} p_2^{\alpha_2^{(2)}} \dots p_t^{\alpha_t^{(2)}} && \longrightarrow && v_2 = \left(\alpha_1^{(2)} \pmod e,\ \alpha_2^{(2)} \pmod e, \dots,\ \alpha_t^{(2)} \pmod e\right) \\
&\dots \\
\mu(M_{t+1}) &= p_1^{\alpha_1^{(t+1)}} p_2^{\alpha_2^{(t+1)}} \dots p_t^{\alpha_t^{(t+1)}} && \longrightarrow && v_{t+1} = \left(\alpha_1^{(t+1)} \pmod e,\ \alpha_2^{(t+1)} \pmod e, \dots,\ \alpha_t^{(t+1)} \pmod e\right)
\end{aligned}
$$

Crucial observation:

- We have $t+1$ vectors in a space of dimension t
- $\Rightarrow$ **They must be linearly dependent!**

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- The vectors are linearly dependent $\Rightarrow$ one of the vectors can be expressed as a linear combination of the others

$$\Rightarrow \quad v_{t+1} \quad = \quad \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t \pmod{e}$$

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- The vectors are linearly dependent $\Rightarrow$ one of the vectors can be expressed as a linear combination of the others

$$\begin{aligned}\Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t \pmod{e} \\ \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t + \gamma e, \text{ for some vector } \gamma\end{aligned}$$

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- The vectors are linearly dependent $\Rightarrow$ one of the vectors can be expressed as a linear combination of the others

$$\begin{aligned} \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t \pmod{e} \\ \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t + \gamma e, \text{ for some vector } \gamma \end{aligned}$$

- Then for i-th coordinate of $v_{t+1} = \left(\alpha_1^{(t+1)} (\text{mod } e),\ \alpha_2^{(t+1)} (\text{mod } e), \ldots,\ \alpha_t^{(t+1)} (\text{mod } e)\right)$:

$$\alpha_i^{(t+1)} = \beta_1 \alpha_i^{(1)} + \beta_2 \alpha_i^{(2)} + \cdots + \beta_t \alpha_i^{(t)} + \gamma_i e$$

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- The vectors are linearly dependent $\Rightarrow$ one of the vectors can be expressed as a linear combination of the others

$$\begin{aligned} \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t \pmod{e} \\ \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t + \gamma e, \text{ for some vector } \gamma \end{aligned}$$

- Then for i-th coordinate of $v_{t+1} = \left(\alpha_1^{(t+1)} \pmod{e},\ \alpha_2^{(t+1)} \pmod{e}, \ldots,\ \alpha_t^{(t+1)} \pmod{e}\right)$:

$$\begin{aligned} \alpha_i^{(t+1)} &= \beta_1 \alpha_i^{(1)} + \beta_2 \alpha_i^{(2)} + \cdots + \beta_t \alpha_i^{(t)} + \gamma_i e \\ p_i^{\alpha_i^{(t+1)}} &= p_i^{\beta_1 \alpha_i^{(1)}} \cdot p_i^{\beta_2 \alpha_i^{(2)}} \cdot \cdots \cdot p_i^{\beta_t \alpha_i^{(t)}} \cdot p_i^{\gamma_i e} \end{aligned}$$

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- The vectors are linearly dependent $\Rightarrow$ one of the vectors can be expressed as a linear combination of the others

$$\begin{aligned} \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t \pmod{e} \\ \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t + \gamma e, \text{ for some vector } \gamma \end{aligned}$$

- Then for i-th coordinate of $v_{t+1} = \left(\alpha_1^{(t+1)} (\text{mod } e),\ \alpha_2^{(t+1)} (\text{mod } e), \ldots,\ \alpha_t^{(t+1)} (\text{mod } e)\right)$:

$$\begin{aligned} \alpha_i^{(t+1)} &= \beta_1 \alpha_i^{(1)} + \beta_2 \alpha_i^{(2)} + \cdots + \beta_t \alpha_i^{(t)} + \gamma_i e \\ p_i^{\alpha_i^{(t+1)}} &= p_i^{\beta_1 \alpha_i^{(1)}} \cdot p_i^{\beta_2 \alpha_i^{(2)}} \cdot \cdots \cdot p_i^{\beta_t \alpha_i^{(t)}} \cdot p_i^{\gamma_i e} \end{aligned}$$

- And then combined:

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- The vectors are linearly dependent $\Rightarrow$ one of the vectors can be expressed as a linear combination of the others

$$\begin{aligned} \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t \pmod{e} \\ \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t + \gamma e, \text{ for some vector } \gamma \end{aligned}$$

- Then for i-th coordinate of $v_{t+1} = \left(\alpha_1^{(t+1)} (\bmod\ e),\ \alpha_2^{(t+1)} (\bmod\ e), \ldots,\ \alpha_t^{(t+1)} (\bmod\ e)\right)$:

$$\begin{aligned} \alpha_i^{(t+1)} &= \beta_1 \alpha_i^{(1)} + \beta_2 \alpha_i^{(2)} + \cdots + \beta_t \alpha_i^{(t)} + \gamma_i e \\ p_i^{\alpha_i^{(t+1)}} &= p_i^{\beta_1 \alpha_i^{(1)}} \cdot p_i^{\beta_2 \alpha_i^{(2)}} \cdot \cdots \cdot p_i^{\beta_t \alpha_i^{(t)}} \cdot p_i^{\gamma_i e} \end{aligned}$$

- And then combined:

$$\prod_{i=1}^{t} p_i^{\alpha_i^{(t+1)}} = \prod_{i=1}^{t} p_i^{\beta_1 \alpha_i^{(1)}} \cdot \prod_{i=1}^{t} p_i^{\beta_2 \alpha_i^{(2)}} \cdot \cdots \cdot \prod_{i=1}^{t} p_i^{\beta_t \alpha_i^{(t)}} \cdot \prod_{i=1}^{t} p_i^{\gamma_i e}$$

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- The vectors are linearly dependent $\Rightarrow$ one of the vectors can be expressed as a linear combination of the others

$$\begin{aligned}\Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t \pmod{e} \\ \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t + \gamma e, \text{ for some vector } \gamma\end{aligned}$$

- Then for i-th coordinate of $v_{t+1} = \left(\alpha_1^{(t+1)} (\text{mod } e),\ \alpha_2^{(t+1)} (\text{mod } e), \ldots,\ \alpha_t^{(t+1)} (\text{mod } e)\right)$:

$$\begin{aligned}\alpha_i^{(t+1)} &= \beta_1 \alpha_i^{(1)} + \beta_2 \alpha_i^{(2)} + \cdots + \beta_t \alpha_i^{(t)} + \gamma_i e \\ p_i^{\alpha_i^{(t+1)}} &= p_i^{\beta_1 \alpha_i^{(1)}} \cdot p_i^{\beta_2 \alpha_i^{(2)}} \cdot \cdots \cdot p_i^{\beta_t \alpha_i^{(t)}} \cdot p_i^{\gamma_i e}\end{aligned}$$

- And then combined:

$$\prod_{i=1}^{t} p_i^{\alpha_i^{(t+1)}} = \prod_{i=1}^{t} p_i^{\beta_1 \alpha_i^{(1)}} \cdot \prod_{i=1}^{t} p_i^{\beta_2 \alpha_i^{(2)}} \cdot \cdots \cdot \prod_{i=1}^{t} p_i^{\beta_t \alpha_i^{(t)}} \cdot \prod_{i=1}^{t} p_i^{\gamma_i e}$$

- Finally:

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- The vectors are linearly dependent $\Rightarrow$ one of the vectors can be expressed as a linear combination of the others

$$\begin{aligned}\Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t \pmod{e} \\ \Rightarrow \quad v_{t+1} &= \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_t v_t + \gamma e, \text{ for some vector } \gamma\end{aligned}$$

- Then for i-th coordinate of $v_{t+1} = \left(\alpha_1^{(t+1)} (\mathrm{mod}\ e),\ \alpha_2^{(t+1)} (\mathrm{mod}\ e), \ldots,\ \alpha_t^{(t+1)} (\mathrm{mod}\ e)\right)$:

$$\begin{aligned}\alpha_i^{(t+1)} &= \beta_1 \alpha_i^{(1)} + \beta_2 \alpha_i^{(2)} + \cdots + \beta_t \alpha_i^{(t)} + \gamma_i e \\ p_i^{\alpha_i^{(t+1)}} &= p_i^{\beta_1 \alpha_i^{(1)}} \cdot p_i^{\beta_2 \alpha_i^{(2)}} \cdot \cdots \cdot p_i^{\beta_t \alpha_i^{(t)}} \cdot p_i^{\gamma_i e}\end{aligned}$$

- And then combined:

$$\prod_{i=1}^t p_i^{\alpha_i^{(t+1)}} = \prod_{i=1}^t p_i^{\beta_1 \alpha_i^{(1)}} \cdot \prod_{i=1}^t p_i^{\beta_2 \alpha_i^{(2)}} \cdot \cdots \cdot \prod_{i=1}^t p_i^{\beta_t \alpha_i^{(t)}} \cdot \prod_{i=1}^t p_i^{\gamma_i e}$$

- Finally:

$$\mu(M_{t+1}) = \mu(M_1)^{\beta_1} \cdot \mu(M_2)^{\beta_2} \cdot \cdots \cdot \mu(M_t)^{\beta_t} \cdot \prod_{i=1}^t p_i^{\gamma_i e}$$

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}

An example walkthrough - Attack on ISO/IEC 9796 (CNS99)

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- From previous slide:

$$\mu(M_{t+1}) \quad = \quad \mu(M_1)^{\beta_1} \cdot \mu(M_2)^{\beta_2} \cdot \dots \cdot \mu(M_t)^{\beta_t} \cdot \prod_{i=1}^{t} p_i^{\gamma_i e}$$

- Ask for signatures $\sigma_1, \dots, \sigma_t$ of the messages $M_1, \dots, M_t$

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- From previous slide:

$$\mu(M_{t+1}) = \mu(M_1)^{\beta_1} \cdot \mu(M_2)^{\beta_2} \cdot \dots \cdot \mu(M_t)^{\beta_t} \cdot \prod_{i=1}^{t} p_i^{\gamma_i e}$$

- Ask for signatures $\sigma_1, \dots, \sigma_t$ of the messages $M_1, \dots, M_t$
- Compute $\beta_1, \dots, \beta_t, \gamma$ using linear algebra

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- From previous slide:

$$\mu(M_{t+1}) \quad = \quad \mu(M_1)^{\beta_1} \cdot \mu(M_2)^{\beta_2} \cdot \cdots \cdot \mu(M_t)^{\beta_t} \cdot \prod_{i=1}^{t} p_i^{\gamma_i e}$$

- Ask for signatures $\sigma_1, \ldots, \sigma_t$ of the messages $M_1, \ldots, M_t$
- Compute $\beta_1, \ldots, \beta_t, \gamma$ using linear algebra
- Compute $\prod_{i=1}^{t} \sigma_i^{\beta_i} \cdot \prod_{i=1}^{t} p_i^{\gamma_i} \ldots$

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- From previous slide:

$$\mu(M_{t+1}) \quad = \quad \mu(M_1)^{\beta_1} \cdot \mu(M_2)^{\beta_2} \cdot \dots \cdot \mu(M_t)^{\beta_t} \cdot \prod_{i=1}^{t} p_i^{\gamma_i e}$$

- Ask for signatures $\sigma_1, \dots, \sigma_t$ of the messages $M_1, \dots, M_t$
- Compute $\beta_1, \dots, \beta_t, \gamma$ using linear algebra
- Compute $\prod_{i=1}^{t} \sigma_i^{\beta_i} \cdot \prod_{i=1}^{t} p_i^{\gamma_i} \dots$ **Why?**

The attack contd.:

- Suppose we want to forge a signature σ_{t+1} on the message M_{t+1}
- From previous slide:

$$\mu(M_{t+1}) \quad = \quad \mu(M_1)^{\beta_1} \cdot \mu(M_2)^{\beta_2} \cdot \dots \cdot \mu(M_t)^{\beta_t} \cdot \prod_{i=1}^{t} p_i^{\gamma_i e}$$

- Ask for signatures $\sigma_1, \dots, \sigma_t$ of the messages $M_1, \dots, M_t$
- Compute $\beta_1, \dots, \beta_t, \gamma$ using linear algebra
- Compute $\prod_{i=1}^{t} \sigma_i^{\beta_i} \cdot \prod_{i=1}^{t} p_i^{\gamma_i} \dots$ **Why?**
- Since $\sigma_i = \mu(M_i)^d \pmod N$:

$$\begin{aligned} \prod_{i=1}^{t} \sigma_i^{\beta_i} \cdot \prod_{i=1}^{t} p_i^{\gamma_i} &= \prod_{i=1}^{t} \mu(M_i)^{d\beta_i} \prod_{i=1}^{t} p_i^{\gamma_i ed} \pmod N \\ &= \left(\prod_{i=1}^{t} \mu(M_i)^{\beta_i}\right)^d \left(\prod_{i=1}^{t} p_i^{\gamma_i e}\right)^d \pmod N \\ &= \mu(M_{t+1})^d \pmod N \end{aligned}$$

- **Voila! We have a forged signature** $\sigma_{t+1} = \mu(M_{t+1})^d \pmod N$ of M_{t+1}

An EUF-CMA secure DSs - generic construction FDH

Trapdoor (one-way) permutation:
$\mathcal{T}$ is a trapdoor permutation if it is easy to compute $\mathcal{T}(\mathsf{pk}, x) = \pi(x)$ for any x in the domain D, but **given** b from the range R it is **computationally hard to find** $a \in D$, such that

$$\mathcal{T}(\mathsf{pk}, a) = b$$

without the knowledge of a trapdoor sk.
When the trapdoor is known, $a = \mathcal{T}(\mathsf{sk}, b) = \pi^{-1}(b)$ is easy to compute.

An EUF-CMA secure DSs - generic construction FDH

Trapdoor (one-way) permutation:
$\mathcal{T}$ is a trapdoor permutation if it is easy to compute $\mathcal{T}(\mathsf{pk}, x) = \pi(x)$ for any x in the domain D, but **given** b from the range R it is **computationally hard to find** $a \in D$, such that

$$\mathcal{T}(\mathsf{pk}, a) = b$$

without the knowledge of a trapdoor sk.
When the trapdoor is known, $a = \mathcal{T}(\mathsf{sk}, b) = \pi^{-1}(b)$ is easy to compute.

We further need:

- a **Full Domain Hash** function $FDH : \{0,1\}^* \to D$ modelled as a random oracle

An EUF-CMA secure DSs - generic construction FDH

Trapdoor (one-way) permutation:
$\mathcal{T}$ is a trapdoor permutation if it is easy to compute $\mathcal{T}(\mathsf{pk}, x) = \pi(x)$ for any x in the domain D, but **given** b from the range R it is **computationally hard to find** $a \in D$, such that

$$\mathcal{T}(\mathsf{pk}, a) = b$$

without the knowledge of a trapdoor sk.
When the trapdoor is known, $a = \mathcal{T}(\mathsf{sk}, b) = \pi^{-1}(b)$ is easy to compute.

We further need:

- a **Full Domain Hash** function $FDH : \{0,1\}^* \rightarrow D$ modelled as a random oracle

Construction of FDH DSs:

KGen: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^k)$ and trapdoor permutation $\mathcal{T}$

Sign: Compute $y = FDH(M)$ and calculate signature $\sigma = \mathcal{T}(\mathsf{sk}, y)$

Vf: Given message and signature pair (M, σ), compute $y' = \mathcal{T}(\mathsf{pk}, \sigma)$ and output $y' \stackrel{?}{=} FDH(M)$

An EUF-CMA secure DSs - RSA-FDH

Construction of RSA-FDH DSs:

KeyGen:

1. Choose two primes p, q s.t. $|p| \approx |q|$
2. Compute $N = pq$ and $\phi(N) = (p-1)(q-1)$
3. Choose a random $e < \phi(N)$, s.t. $\gcd(e, \phi(N)) = 1$
4. Compute d such that $ed = 1 \pmod{\phi(N)}$
5. Output public key $\mathsf{pk} = (N, e)$ and private key $\mathsf{sk} = d$

Sign: Given message M, compute signature by $\sigma \leftarrow FDH(M)^d \pmod{N}$

Verify: To verify the message - signature pair (M, σ) compute $h' \leftarrow \sigma^e \pmod{N}$

If $h' = FDH(M)$ output *Accept*

Construction of RSA-FDH DSs:

KeyGen:

1. Choose two primes p, q s.t. $|p| \approx |q|$
2. Compute $N = pq$ and $\phi(N) = (p-1)(q-1)$
3. Choose a random $e < \phi(N)$, s.t. $\gcd(e, \phi(N)) = 1$
4. Compute d such that $ed = 1 \pmod{\phi(N)}$
5. Output public key $\mathsf{pk} = (N, e)$ and private key $\mathsf{sk} = d$

Sign: Given message M, compute signature by $\sigma \leftarrow FDH(M)^d \pmod{N}$

Verify: To verify the message - signature pair (M, σ) compute $h' \leftarrow \sigma^e \pmod{N}$

If $h' = FDH(M)$ output *Accept*

RSA trapdoor permutation:

$\mathcal{T}(\mathsf{pk}, x) = \pi(x) = x^e \pmod{N}$ and $\mathcal{T}(\mathsf{sk}, y) = y^d \pmod{N}$.

It is computationally hard to find $\pi^{-1}(y)$ without the knowledge of d if the RSA assumption holds.

RSA assumption: It is hard to find x, given $y = x^e \pmod{N}, e$ and N.

An EUF-CMA secure DSs - RSA-FDH

EUF-CMA security: If the RSA assumption holds then RSA-FDH DSs is EUF-CMA secure in the random oracle model (ROM).

EUF-CMA security: If the RSA assumption holds then RSA-FDH DSs is EUF-CMA secure in the random oracle model (ROM).

Sketch of proof: Suppose $\mathcal{A}$ (the forger) has non-negligible advantage against DSs in an EUF-CMA game (can (t, ϵ) break it). We construct simulator $\mathcal{B}$ (the inverter) that inverts the trapdoor permutation $\mathcal{T}$ with non-negligible probability.

An EUF-CMA secure DSs - RSA-FDH

EUF-CMA security: If the RSA assumption holds then RSA-FDH DSs is EUF-CMA secure in the random oracle model (ROM).

Sketch of proof: Suppose $\mathcal{A}$ (the forger) has non-negligible advantage against DSs in an EUF-CMA game (can (t, ϵ) break it). We construct simulator $\mathcal{B}$ (the inverter) that inverts the trapdoor permutation $\mathcal{T}$ with non-negligible probability.

Setup: $\mathcal{B}$ is given an RSA instance (N, e, y) where $y \in \mathbb{Z}_N^*$. His goal is to find $x = \pi^{-1}(y)$.

$\mathcal{B}$ sets $\mathrm{pk} = (N, e)$ as the public key that $\mathcal{A}$ attacks.

An EUF-CMA secure DSs - RSA-FDH

EUF-CMA security: If the RSA assumption holds then RSA-FDH DSs is EUF-CMA secure in the random oracle model (ROM).

Sketch of proof: Suppose $\mathcal{A}$ (the forger) has non-negligible advantage against DSs in an EUF-CMA game (can (t, ϵ) break it). We construct simulator $\mathcal{B}$ (the inverter) that inverts the trapdoor permutation $\mathcal{T}$ with non-negligible probability.

Setup: $\mathcal{B}$ is given an RSA instance (N, e, y) where $y \in \mathbb{Z}_N^*$. His goal is to find $x = \pi^{-1}(y)$.
$\mathcal{B}$ sets $\mathrm{pk} = (N, e)$ as the public key that $\mathcal{A}$ attacks.

FDH-queries: $\mathcal{A}$ asks for signatures of messages $\Rightarrow$ $\mathcal{A}$ makes hash queries to FDH

- $\mathcal{B}$ simulates FDH by maintaining a list FDH_L of queries (M_i, r_i, h_i)
- i-th query M_i: If M_i in list, answer with h_i; if not, pick randomly $r_i \in \mathbb{Z}_N^*$, set $h_i = r_i^e \pmod N$ with probability p and $h_i = y \cdot r_i^e \pmod N$ with probability $1 - p$. Add (M_i, r_i, h_i) to list FDH_L.

An EUF-CMA secure DSs - RSA-FDH

EUF-CMA security: If the RSA assumption holds then RSA-FDH DSs is EUF-CMA secure in the random oracle model (ROM).

Sketch of proof: Suppose $\mathcal{A}$ (the forger) has non-negligible advantage against DSs in an EUF-CMA game (can (t, ϵ) break it). We construct simulator $\mathcal{B}$ (the inverter) that inverts the trapdoor permutation $\mathcal{T}$ with non-negligible probability.

Setup: $\mathcal{B}$ is given an RSA instance (N, e, y) where $y \in \mathbb{Z}_N^*$. His goal is to find $x = \pi^{-1}(y)$.

$\mathcal{B}$ sets $\mathrm{pk} = (N, e)$ as the public key that $\mathcal{A}$ attacks.

FDH-queries: $\mathcal{A}$ asks for signatures of messages $\Rightarrow$ $\mathcal{A}$ makes hash queries to FDH

- $\mathcal{B}$ simulates FDH by maintaining a list FDH_L of queries (M_i, r_i, h_i)
- i-th query M_i: If M_i in list, answer with h_i; if not, pick randomly $r_i \in \mathbb{Z}_N^*$, set $h_i = r_i^e \pmod N$ with probability p and $h_i = y \cdot r_i^e \pmod N$ with probability $1 - p$. Add (M_i, r_i, h_i) to list FDH_L.

Signature-queries: $\mathcal{A}$ asks for signatures of messages

- When $\mathcal{A}$ queries M: $\mathcal{A}$ has already queried the hash oracle FDH, so $M = M_i$ is in the list, for some i. If $h_i = r_i^e \pmod N$, then $\mathcal{B}$ returns r_i as the signature. Otherwise, outputs $\perp$ and halts (it has failed to invert the trapdoor).

An EUF-CMA secure DSs - RSA-FDH

Sketch of proof, contd.:

Forgery: $\mathcal{A}$ outputs a forgery (M^*, σ^*). We assume that $\mathcal{A}$ has queried the FDH oracle for M^*, i.e. it is in the list FDH_L for some i. (If not, $\mathcal{B}$ just makes the query itself.)

- If σ^* is valid, then $\sigma^* = h_i^d$

Sketch of proof, contd.:

Forgery: $\mathcal{A}$ outputs a forgery (M^*, σ^*). We assume that $\mathcal{A}$ has queried the FDH oracle for M^*, i.e. it is in the list FDH_L for some i. (If not, $\mathcal{B}$ just makes the query itself.)

- If σ^* is valid, then $\sigma^* = h_i^d$
- Then, for $h_i = y \cdot r_i^e \pmod N$ we have: $\sigma^* = h_i^d = y^d \cdot r_i \pmod N$, so $y^d = \sigma^*/r_i$

An EUF-CMA secure DSs - RSA-FDH

Sketch of proof, contd.:

Forgery: $\mathcal{A}$ outputs a forgery (M^*, σ^*). We assume that $\mathcal{A}$ has queried the FDH oracle for M^*, i.e. it is in the list FDH_L for some i. (If not, $\mathcal{B}$ just makes the query itself.)

- If σ^* is valid, then $\sigma^* = h_i^d$
- Then, for $h_i = y \cdot r_i^e \pmod{N}$ we have: $\sigma^* = h_i^d = y^d \cdot r_i \pmod{N}$, so $y^d = \sigma^* / r_i$

Output: If $h_i = y \cdot r_i^e \pmod{N}$, the $\mathcal{B}$ outputs σ^*/r_i as the inverse of y. Otherwise, outputs $\perp$ and halts.

An EUF-CMA secure DSs - RSA-FDH

Sketch of proof, contd.:

Forgery: $\mathcal{A}$ outputs a forgery (M^*, σ^*). We assume that $\mathcal{A}$ has queried the FDH oracle for M^*, i.e. it is in the list FDH_L for some i. (If not, $\mathcal{B}$ just makes the query itself.)

- If σ^* is valid, then $\sigma^* = h_i^d$
- Then, for $h_i = y \cdot r_i^e \pmod{N}$ we have: $\sigma^* = h_i^d = y^d \cdot r_i \pmod{N}$, so $y^d = \sigma^* / r_i$

Output: If $h_i = y \cdot r_i^e \pmod{N}$, the $\mathcal{B}$ outputs σ^* / r_i as the inverse of y. Otherwise, outputs $\perp$ and halts.

Analysis: Probability that $\mathcal{B}$ outputs something (different from $\perp$):

- $\mathcal{B}$ answers all signature queries: $p^{q_{sig}}$, where q_{sig} - number of signature queries

Sketch of proof, contd.:

Forgery: $\mathcal{A}$ outputs a forgery (M^*, σ^*). We assume that $\mathcal{A}$ has queried the FDH oracle for M^*, i.e. it is in the list FDH_L for some i. (If not, $\mathcal{B}$ just makes the query itself.)

- If σ^* is valid, then $\sigma^* = h_i^d$
- Then, for $h_i = y \cdot r_i^e \pmod N$ we have: $\sigma^* = h_i^d = y^d \cdot r_i \pmod N$, so $y^d = \sigma^* / r_i$

Output: If $h_i = y \cdot r_i^e \pmod N$, the $\mathcal{B}$ outputs σ^*/r_i as the inverse of y. Otherwise, outputs $\perp$ and halts.

Analysis: Probability that $\mathcal{B}$ outputs something (different from $\perp$):

- $\mathcal{B}$ answers all signature queries: $p^{q_{sig}}$, where q_{sig} - number of signature queries
- then $\mathcal{B}$ outputs inverse of y: $1 - p$
- Total $\alpha(p) = p^{q_{sig}}(1-p)$, maximum obtained for $p_{max} = 1 - \frac{1}{q_{sig}+1}$ (**How?**)

An EUF-CMA secure DSs - RSA-FDH

Sketch of proof, contd.:

Forgery: $\mathcal{A}$ outputs a forgery (M^*, σ^*). We assume that $\mathcal{A}$ has queried the FDH oracle for M^*, i.e. it is in the list FDH_L for some i. (If not, $\mathcal{B}$ just makes the query itself.)

- If σ^* is valid, then $\sigma^* = h_i^d$
- Then, for $h_i = y \cdot r_i^e \pmod{N}$ we have: $\sigma^* = h_i^d = y^d \cdot r_i \pmod{N}$, so $y^d = \sigma^*/r_i$

Output: If $h_i = y \cdot r_i^e \pmod{N}$, the $\mathcal{B}$ outputs σ^*/r_i as the inverse of y. Otherwise, outputs $\perp$ and halts.

Analysis: Probability that $\mathcal{B}$ outputs something (different from $\perp$):

- $\mathcal{B}$ answers all signature queries: $p^{q_{sig}}$, where q_{sig} - number of signature queries
- then $\mathcal{B}$ outputs inverse of y: $1 - p$
- Total $\alpha(p) = p^{q_{sig}}(1-p)$, maximum obtained for $p_{max} = 1 - \frac{1}{q_{sig}+1}$ (**How?**)
- Success probability: $\epsilon' = \alpha(p_{max})\epsilon = (1 - \frac{1}{q_{sig}+1})^{q_{sig}+1} \frac{1}{q_{sig}} \epsilon \to \frac{1}{e \cdot q_{sig}} \epsilon$
- Cost: $t + T_s$, T_s - cost of simulation
- $\Rightarrow$ The adversary $\mathcal{B}$ $(t + T_s, \epsilon/eq_{sig})$ inverts the RSA trapdoor

Use of RSA signatures in practice

- Difference is mainly in the function μ

Use of RSA signatures in practice

- Difference is mainly in the function μ
- Focus on resistance to multiplicative forgery (but choices many times ad-hoc)

Ad-Hoc Designs:

Use of RSA signatures in practice

- Difference is mainly in the function μ
- Focus on resistance to multiplicative forgery (but choices many times ad-hoc)

Ad-Hoc Designs:

ISO 9796-1, ISO 9796-2

- Ad-hoc padding scheme (no proof) ($\mu(M) = 6a||m[1]||Hash(m)||bc$ in ISO 9796-2)
- Broken by extension of Desmedt-Odlyzko attack [CNS99] (see previous slides)
- Amended several times: increase of hash length

Use of RSA signatures in practice

- Difference is mainly in the function μ
- Focus on resistance to multiplicative forgery (but choices many times ad-hoc)

Ad-Hoc Designs:

ISO 9796-1, ISO 9796-2

- Ad-hoc padding scheme (no proof) ($\mu(M) = 6a||m[1]||Hash(m)||bc$ in ISO 9796-2)
- Broken by extension of Desmedt-Odlyzko attack [CNS99] (see previous slides)
- Amended several times: increase of hash length

ANSI X 9. 31 (Digital Signatures Using Reversible PKC for the Financial Services Industry, 1998)

- Ad-hoc padding scheme (no proof) ($\mu(M) = 6bbb\ldots bbba||Hash(M)||3xcc$)
- **Several other standards**: IEEE P 1363, ISO/IEC 14888 -3, US NIST FIPS 186 -1

Use of RSA signatures in practice

- Difference is mainly in the function μ
- Focus on resistance to multiplicative forgery (but choices many times ad-hoc)

Ad-Hoc Designs:

ISO 9796-1, ISO 9796-2

- Ad-hoc padding scheme (no proof) ($\mu(M) = 6a||m[1]||Hash(m)||bc$ in ISO 9796-2)
- Broken by extension of Desmedt-Odlyzko attack [CNS99] (see previous slides)
- Amended several times: increase of hash length

ANSI X 9. 31 (Digital Signatures Using Reversible PKC for the Financial Services Industry, 1998)

- Ad-hoc padding scheme (no proof) ($\mu(M) = 6bbb\ldots bbba||Hash(M)||3xcc$)
- **Several other standards**: IEEE P 1363, ISO/IEC 14888 -3, US NIST FIPS 186 -1

PKCS #1 v1.5

- Ad-hoc padding scheme (no proof) ($\mu(M) = 0001ff\ldots ff00||Hash.Alg.ID||Hash(M)$)
- IEEE P 1363a, RSA tokens, Gemalto tokens, ID cards, certificates

Use of RSA signatures in practice

- Difference is mainly in the function μ
- Focus on resistance to multiplicative forgery (but choices many times ad-hoc)

Ad-Hoc Designs:

ISO 9796-1, ISO 9796-2

- Ad-hoc padding scheme (no proof) ($\mu(M) = 6a||m[1]||Hash(m)||bc$ in ISO 9796-2)
- Broken by extension of Desmedt-Odlyzko attack [CNS99] (see previous slides)
- Amended several times: increase of hash length

ANSI X 9. 31 (Digital Signatures Using Reversible PKC for the Financial Services Industry, 1998)

- Ad-hoc padding scheme (no proof) ($\mu(M) = 6bbb\ldots bbba||Hash(M)||3xcc$)
- **Several other standards**: IEEE P 1363, ISO/IEC 14888 -3, US NIST FIPS 186 -1

PKCS #1 v1.5

- Ad-hoc padding scheme (no proof) ($\mu(M) = 0001ff\ldots ff00||Hash.Alg.ID||Hash(M)$)
- IEEE P 1363a, RSA tokens, Gemalto tokens, ID cards, certificates

Use of RSA signatures in practice

Provably secure Designs:

Use of RSA signatures in practice

Provably secure Designs:

RSA-FDH (Bellare and Rogaway, ACM CCS '93)

- Provably secure in the ROM (see proof in previous slides)
- deterministic
- **Standards**: IEEE P1363a

Use of RSA signatures in practice

Provably secure Designs:

RSA-FDH (Bellare and Rogaway, ACM CCS '93)

- Provably secure in the ROM (see proof in previous slides)
- deterministic
- **Standards**: IEEE P1363a

RSA-PSS (Probabilistic signature scheme - Bellare and Rogaway, Eurocrypt'96)

- Provably secure in the ROM - tight reduction from RSA problem
- randomized version of RSA-FDH
- $\mu(M) = 00||Hash(salt, M)||G(Hash(salt, M)) \oplus [salt||00 \ldots 00]$
- **Standards**: IEEE P1363a and PKCS#1 v2.1

Use of RSA signatures in practice

Provably secure Designs:

RSA-FDH (Bellare and Rogaway, ACM CCS '93)

- Provably secure in the ROM (see proof in previous slides)
- deterministic
- **Standards**: IEEE P1363a

RSA-PSS (Probabilistic signature scheme - Bellare and Rogaway, Eurocrypt'96)

- Provably secure in the ROM - tight reduction from RSA problem
- randomized version of RSA-FDH
- $\mu(M) = 00||Hash(salt, M)||G(Hash(salt, M)) \oplus [salt||00\ldots00]$
- **Standards**: IEEE P1363a and PKCS#1 v2.1

Practical threats

AUSTRALIA EDITION

itnews

NEWS | GOVERNMENT | SECURITY | RESOURCES | PODCAST | STATE OF IT | DIGITAL NATION | BENCHMARK AWARDS | LOG IN | SUBSCRIBE

RSA tokens 'broken' in 13 minutes

By Dan Raywood
Jun 27 2012
12:52PM

0 Comments

The fragility of authentication tokens against established attack vectors have been detailed.

The group are to present a paper on the subject at the Crypto 2012 conference in August in Santa Barbara, California. They also confirmed that the SecurID 800 and other tokens can be broken.

The paper authored by Team Prosecco (Romain Bardou, Riccardo Focardi, Yusuke Kawamoto, Lorenzo Simionato, Graham Steel and Joe-Kai Tsay) detailed a demonstration on how to exploit the encrypted key import functions of a variety of different cryptographic devices to reveal the imported key.

RELATED ARTICLES

ASD says quantum no immediate threat to

Is that all?

- RSA signatures are used a lot!

Is that all?

- RSA signatures are used a lot!
- Easy to implement, with very fast verification algorithm

Is that all?

- RSA signatures are used a lot!
- Easy to implement, with very fast verification algorithm
- Easy to implement wrongly

Is that all?

- RSA signatures are used a lot!
- Easy to implement, with very fast verification algorithm
- Easy to implement wrongly
- Major design feature: **Signature from trapdoor permutation**

Is that all?

- RSA signatures are used a lot!
- Easy to implement, with very fast verification algorithm
- Easy to implement wrongly
- Major design feature: **Signature from trapdoor permutation**
- The rest of the landscape? Signatures from other trapdoor permutations?

Is that all?

- RSA signatures are used a lot!
- Easy to implement, with very fast verification algorithm
- Easy to implement wrongly
- Major design feature: **Signature from trapdoor permutation**
- The rest of the landscape? Signatures from other trapdoor permutations?
- **Luckily no!**

Is that all?

- RSA signatures are used a lot!
- Easy to implement, with very fast verification algorithm
- Easy to implement wrongly
- Major design feature: **Signature from trapdoor permutation**
- The rest of the landscape? Signatures from other trapdoor permutations?
- **Luckily no!**

Modern signature designs:

- **Fiat-Shamir signatures**

Is that all?

- RSA signatures are used a lot!
- Easy to implement, with very fast verification algorithm
- Easy to implement wrongly
- Major design feature: **Signature from trapdoor permutation**
- The rest of the landscape? Signatures from other trapdoor permutations?
- **Luckily no!**

Modern signature designs:

- **Fiat-Shamir signatures**
  - Schnorr signatures, many modern post-quantum signatures
  - Similarities with DSA, ECDSA
- **Hash-based signatures**

Is that all?

- RSA signatures are used a lot!
- Easy to implement, with very fast verification algorithm
- Easy to implement wrongly
- Major design feature: **Signature from trapdoor permutation**
- The rest of the landscape? Signatures from other trapdoor permutations?
- **Luckily no!**

Modern signature designs:

- **Fiat-Shamir signatures**
  - Schnorr signatures, many modern post-quantum signatures
  - Similarities with DSA, ECDSA
- **Hash-based signatures**

- We will talk about these in the next lectures...

Summary

Today:

- Key Encapsulation Mechanisms
- Digital Signatures from trapdoor permutations
- Security of Public Key Encryption (contd.)

Summary

Today:

- Key Encapsulation Mechanisms
- Digital Signatures from trapdoor permutations
- Security of Public Key Encryption (contd.)

Next time:

- Commitment schemes
- Zero-Knowledge protocols
- Sigma protocols and identification schemes
- Fiat-Shamir transform