Lecture 9 – Public-Key Encryption

COSC-466 Applied Cryptography

Adam O'Neill

Adapted from

http://cseweb.ucsd.edu/~mihir/cse107/

Recall Symmetric-Key Crypto

- In this setting, if Alice wants to communicate secure with Bob they need a shared key K_{AB}.

Recall Symmetric-Key Crypto

- In this setting, if Alice wants to communicate secure with Bob they need a shared key K_{AB}.
- If Alice wants to also communicate with Charlie they need a shared key K_{AC}.

Recall Symmetric-Key Crypto

- In this setting, if Alice wants to communicate secure with Bob they need a shared key K_{AB}.
- If Alice wants to also communicate with Charlie they need a shared key K_{AC}.
- If Alice generates K_{AB} and K_{AC} they must be communicated to Bob and Charlie over secure channels. How can this be done?

Public-Key Crypto

- Alice has a secret key that is shared with nobody, and a public key that is shared with everybody.

Public-Key Crypto

- Alice has a secret key that is shared with nobody, and a public key that is shared with everybody.
- Anyone can use Alice's public key to send her a private message.

Public-Key Crypto

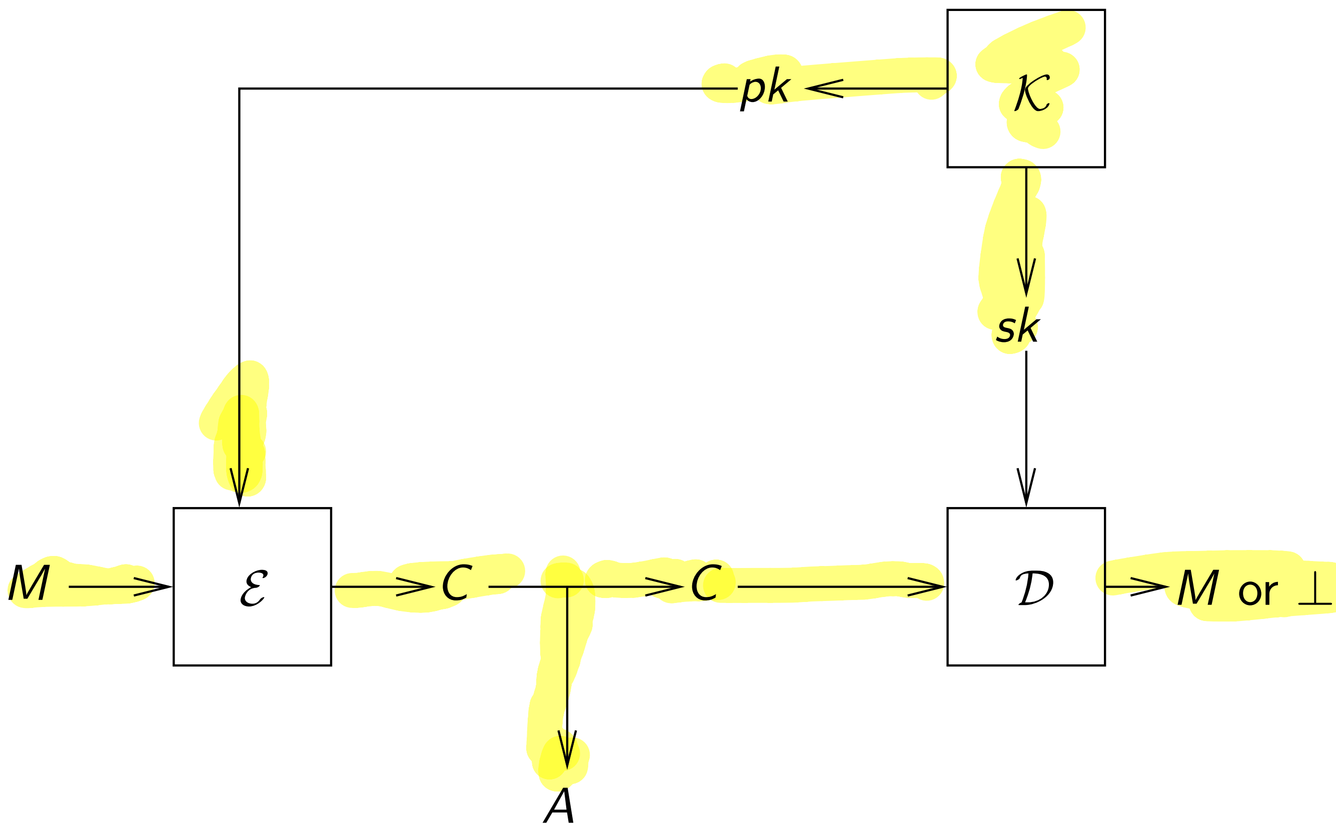
- Alice has a secret key that is shared with nobody, and a public key that is shared with everybody.
- Anyone can use Alice’s public key to send her a private message.
- Public key is like a phone number: Anyone can look it up in a phone book.

Public-Key Crypto

- Alice has a secret key that is shared with nobody, and a public key that is shared with everybody.
- Anyone can use Alice’s public key to send her a private message.
- Public key is like a phone number: Anyone can look it up in a phone book.
- Senders don’t need secrets; there are no shared secrets

Syntax and Correctness of PKE

A public-key (or asymmetric) encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms, where



Code Obfuscation Perspective

On input m, r

output

$r, AES_K(r) \oplus m$

Diffie-Hellman

PKE?

Correctness

Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an asymmetric encryption scheme. The correct decryption requirement is that

$$\Pr[\mathcal{D}(sk, \mathcal{E}(pk, M)) = M] = 1$$

for all (pk, sk) that may be output by $\mathcal{K}$ and all messages M in the *message space* of $\mathcal{AE}$. The probability is over the random choices of $\mathcal{E}$.

This simply says that decryption correctly reverses encryption to recover the message that was encrypted. When we specify schemes, we indicate what is the message space.

How It Works

Step 1: Key generation
Alice locally computers $(pk, sk) \xleftarrow{\$} \mathcal{K}$ and stores sk.

Step 2: Alice enables any prospective sender to get pk.

Step 3: The sender encrypts under *pk* and Alice decrypts under sk.

We don't require privacy of pk but we do require authenticity: the sender should be assured pk is really Alice's key and not someone else's. One could

- Put public keys in a trusted but public "phone book", say a cryptographic DNS.
- Use certificates as we will see later.

IND-CPA

Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a PKE scheme and A an adversary.

Game $\mathrm{Left}_{\mathcal{AE}}$	Game $\mathrm{Right}_{\mathcal{AE}}$
procedure Initialize	**procedure Initialize**
$(pk, sk) \xleftarrow{\$} \mathcal{K}$; $\text{return } pk$	$(pk, sk) \xleftarrow{\$} \mathcal{K}$; $\text{return } pk$
procedure LR(M_0, M_1)	**procedure LR**(M_0, M_1)
Return $C \xleftarrow{\$} \mathcal{E}_{pk}(M_0)$	Return $C \xleftarrow{\$} \mathcal{E}_{pk}(M_1)$

Associated to $\mathcal{AE}, A$ are the probabilities

$$\Pr\left[\mathrm{Left}_{\mathcal{AE}}^{A} \Rightarrow 1\right] \qquad \Big| \qquad \Pr\left[\mathrm{Right}_{\mathcal{AE}}^{A} \Rightarrow 1\right]$$

that A outputs 1 in each world. The ind-cpa advantage of A is

$$\mathbf{Adv}_{\mathcal{AE}}^{\mathrm{ind\text{-}cpa}}(A) = \Pr\left[\mathrm{Right}_{\mathcal{AE}}^{A} \Rightarrow 1\right] - \Pr\left[\mathrm{Left}_{\mathcal{AE}}^{A} \Rightarrow 1\right]$$

Explanations

The "return pk" statement in **Initialize** means the adversary A gets the public key pk as input. It does not get sk.

It can call **LR** with any equal-length messages M_0, M_1 of its choice to get back an encryption $C \xleftarrow{\$} \mathcal{E}_{pk}(M_b)$ of M_b under sk, where $b = 0$ in game $\mathrm{Left}_{\mathcal{AE}}$ and $b = 1$ in game $\mathrm{Right}_{\mathcal{AE}}$. Notation indicates encryption algorithm may be randomized.

A is not allowed to call **LR** with messages M_0, M_1 of unequal length. Any such A is considered invalid and its advantage is undefined or 0.

It outputs a bit, and wins if this bit equals b.

How to Build a Scheme?

We would like security to result from the hardness of computing discrete logarithms.

Let the receiver's public key be g where $G = \langle g \rangle$ is a cyclic group. Let's let the encryption of x be g^x. Then

$$\underbrace{g^x}_{\mathcal{E}_g(x)} \xrightarrow{\text{hard}} x$$

so to recover x, adversary must compute discrete logarithms, and we know it can't, so are we done?

Problem: Legitimate receiver needs to compute discrete logarithm to decrypt too! But decryption needs to be feasible.

Above, receiver has no secret key!

A More Basic Problem: Key Exchange

The following are assumed to be public: A large prime p and a generator g of $\mathbf{Z}_p^*$.

Alice		**Bob**
$x \xleftarrow{\$} \mathbf{Z}_{p-1}$; $X \leftarrow g^x \bmod p$		$\boxed{g^y}$
$\boxed{\begin{matrix} g^{yx_1} \\ \vdots \\ g^{y^i x_n} \end{matrix}}$	$\xrightarrow{X}$	
		$y \xleftarrow{\$} \mathbf{Z}_{p-1}$; $Y \leftarrow g^y \bmod p$
	$\xleftarrow{Y}$	
$K_A \leftarrow Y^x \bmod p$		$K_B \leftarrow X^y \bmod p$

- $Y^x = (g^y)^x = g^{xy} = (g^x)^y = X^y$ modulo p, so $K_A = K_B$
- Adversary is faced with the CDH problem.

Key Exchange to PKE

We can turn DH key exchange into a public key encryption scheme via

- Let Alice have public key g^x and secret key x
- If Bob wants to encrypt M for Alice, he
  - Picks y and sends g^y to Alice
  - Encrypts M under $g^{xy} = (g^x)^y$ and sends ciphertext to Alice.
- But Alice can recompute $g^{xy} = (g^y)^x$ because
  - g^y is in the received ciphertext
  - x is her secret key

Thus she can decrypt and adversary is still faced with CDH .

Diffie-Hellman Integrated Encryption Scheme

DHIES

$H(\cancel{g^{xy}}) \oplus (M, g^y) \to \cancel{\ } \oplus H(g^{xy})$

ECIES

Let $G = \langle g \rangle$ be a cyclic group of order m and H: $G \to \{0,1\}^k$ a (public) hash function. The DHIES PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined for messages $M \in \{0,1\}^k$ via

(1) convert unpredictability to randomness → 2 uses of H here

(2) length mis-match

Alg $\mathcal{K}$	**Alg** $\mathcal{E}_X(M)$	**Alg** $\mathcal{D}_x(Y, W)$
$x \xleftarrow{\$} \mathbf{Z}_m$	$y \xleftarrow{\$} \mathbf{Z}_m$; $Y \leftarrow g^y$	$K \leftarrow Y^x$
$X \leftarrow g^x$	$K \leftarrow X^y$	$M \leftarrow H(K) \oplus W$
return (X, x)	$W \leftarrow \boxed{H(K) \oplus M}$	return M
	return (Y, W)	

$\boxed{W \leftarrow AE_{H(K)}(M)}$

Correct decryption is assured because $K = X^y = g^{xy} = Y^x$

Note: This is a simplified version of the actual scheme.

ECIES: 160 + |M| ctxt length for 80-bit sec.

DHIES: 1024 + |M|

Security of DHIES

The DHIES scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ associated to cyclic group $G = \langle g \rangle$ and (public) hash function H can be proven IND-CPA assuming

- CDH is hard in G, and
- H is a "random oracle," meaning a "perfect" hash function.

In practice, $H(K)$ could be the first k bits of the sequence

$$\text{SHA256}(0^8 \| K) \| \text{SHA256}(0^7 1 \| K) \| \cdots$$

ECIES

The DHIES scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ associated to cyclic group $G = \langle g \rangle$ and (public) hash function H can be proven IND-CPA assuming

- CDH is hard in G, and
- H is a "random oracle," meaning a "perfect" hash function.

In practice, $H(K)$ could be the first k bits of the sequence

$$\text{SHA256}(0^8 \| K) \| \text{SHA256}(0^7 1 \| K) \| \cdots$$

DHIES in the case $G = \langle g \rangle$ is an appropriate elliptic curve group.

ECIES features (for 80 bit security)

Operation	Cost
encryption	2 160-bit exponentiation
decryption	1 160-bit exp.

ciphertext expansion =
length of ciphertext - length of plaintext

= 160 bits

Nowadays need
128-bit security

$\Rightarrow$ 256-bit group elements

RSA Math

Recall that $\varphi(N) = |\mathbf{Z}_N^*|$.

Claim: Suppose $e, d \in \mathbf{Z}_{\varphi(N)}^*$ satisfy $ed \equiv 1 \pmod{\varphi(N)}$. Then for any $x \in \mathbf{Z}_N^*$ we have

$$(x^e)^d \equiv x \pmod{N}$$

Proof:

$$(x^e)^d \equiv x^{ed \mod \varphi(N)} \equiv x^1 \equiv x$$

modulo N

The RSA function

$N = p \cdot q$
p, q primes

A modulus N and encryption exponent e define the RSA function $f : \mathbf{Z}_N^* \to \mathbf{Z}_N^*$ defined by

$$f(x) = x^e \mod N$$

$RSA_{N,e}(x) = x^e \bmod N$

for all $x \in \mathbf{Z}_N^*$.

A value $d \in Z_{\varphi(N)}^*$ satisfying $ed \equiv 1 \pmod{\varphi(N)}$ is called a decryption exponent.

Claim: The RSA function $f : \mathbf{Z}_N^* \to \mathbf{Z}_N^*$ is a permutation with inverse $f^{-1} : \mathbf{Z}_N^* \to \mathbf{Z}_N^*$ given by

$$f^{-1}(y) = y^d \mod N$$

Proof: For all $x \in \mathbf{Z}_N^*$ we have

$$f^{-1}(f(x)) \equiv (x^e)^d \equiv x \pmod{N}$$

by previous claim.

Example

Let $N = 15$. So

$$\begin{aligned}
\mathbf{Z}_N^* &= \{1, 2, 4, 7, 8, 11, 13, 14\} \\
\varphi(N) &= 8 \\
\mathbf{Z}_{\varphi(N)}^* &= \{1, 3, 5, 7\}
\end{aligned}$$

Let $e = 3$ and $d = 3$. Then

$$ed \equiv 9 \equiv 1 \pmod{8}$$

Let

$$\begin{aligned}
f(x) &= x^3 \bmod 15 \\
g(y) &= y^3 \bmod 15
\end{aligned}$$

x	$f(x)$	$g(f(x))$
1	1	1
2	8	2
4	4	4
7	13	7
8	2	8
11	11	11
13	7	13
14	14	14

RSA usage

Baby RSA
encryption
we will see
"plain RSA"

- $pk = N, e; \ sk = N, d$
- $\mathcal{E}_{pk}(x) = x^e \mod N = f(x)$
- $\mathcal{D}_{sk}(y) = y^d \mod N = f^{-1}(y)$

Security will rely on it being hard to compute f^{-1} without knowing d.

RSA is a trapdoor, one-way permutation:

- Easy to invert given trapdoor d
- Hard to invert given only N, e

hardness
assumption
on RSA.

RSA generators

There can be many possible RSA generators

An RSA generator with security parameter k is an algorithm $\mathcal{K}_{rsa}$ that returns N, p, q, e, d satisfying

key length

- p, q are distinct odd primes
- $N = pq$ and is called the (RSA) modulus
- $|N| = k$, meaning $2^{k-1} \leq N \leq 2^k$
- $e \in \mathbf{Z}^*_{\varphi(N)}$ is called the encryption exponent
- $d \in \mathbf{Z}^*_{\varphi(N)}$ is called the decryption exponent
- $ed \equiv 1 \pmod{\varphi(N)}$

Example: p, q random

e: chosen at random

Next…

- Building RSA generators
- Basic RSA security
- Encryption with RSA

A formula

Fact: Suppose $N = pq$ for distinct primes p and q. Then

$$\boxed{\varphi(N) = (p-1)(q-1)\ .}$$

Example: Let $N = 15 = 3 \cdot 5$. Then the Fact says that

$$\varphi(15) = (3-1)(5-1) = 8$$

. As a check, $\mathbf{Z}_{15}^{*} = \{1, 2, 4, 7, 8, 11, 13, 14\}$ indeed has size 8.

The general formula

Fact: Suppose $N \geq 1$ factors as

$$N = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_n^{\alpha_n}$$

where $p_1 < p_2 < \ldots < p_n$ are primes and $\alpha_1, \ldots, \alpha_n \geq 1$ are integers. Then

$= 1$ in prev. case $N = pq$

$$\varphi(N) = p_1^{\alpha_1 - 1}(p_1 - 1) \cdot p_2^{\alpha_2 - 1}(p_2 - 1) \cdot \ldots \cdot p_n^{\alpha_n - 1}(p_n - 1) \,.$$

Note prior Fact is a special case of the above. (Make sure you understand why!)

Example: Let $N = 45 = 3^2 \cdot 5^1$. Then the Fact says that

$$\varphi(45) = 3^1(3-1) \cdot 5^0(5-1) = 24$$

Recall

Given $\varphi(N)$ and $e \in \mathbf{Z}^*_{\varphi(N)}$, we can compute $d \in \mathbf{Z}^*_{\varphi(N)}$ satisfying $ed \equiv 1 \pmod{\varphi(N)}$ via

$$d \leftarrow \text{MOD-INV}(e, \varphi(N)).$$

$e'e + \varphi(N)\varphi(N)' = 1$

calls EXT-GCD

We have algorithms to efficiently test whether a number is prime, and a random number has a pretty good chance of being a prime.

Building RSA generators

$e = 2^{k} - 1$ 99%

Say we wish to have $e = 3$ (for efficiency). The generator $\mathcal{K}^3_{rsa}$ with (even) security parameter k:

want modulus length k

repeat
  $p, q \xleftarrow{\$} \{2^{k/2-1}, \ldots, 2^{k/2} - 1\}$; $N \leftarrow pq$; $M \leftarrow (p-1)(q-1)$
until
  $N \geq 2^{k-1}$ and p, q are prime and $gcd(e, M) = 1$
$d \leftarrow \text{MOD-INV}(e, M)$
return N, p, q, e, d

e fixed

p, q random

e relatively prime to

$\varphi(N) = (p-1) \cdot (q-1)$

One-wayness of RSA

RSA generator

The following should be hard:

Given: N, e, y where $y = f(x) = x^e \mod N$

Find: x

Formalism picks x at random and generates N, e via an RSA generator.

One-wayness of RSA formally

Let $\mathcal{K}_{\text{rsa}}$ be a RSA generator and I an adversary.

Game $\mathrm{OW}_{\mathcal{K}_{\text{rsa}}}$	
procedure Initialize $(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$ $x \xleftarrow{\$} \mathbf{Z}_N^*$; $y \leftarrow x^e \mod N$ return N, e, y	**procedure Finalize**(x') return $(x = x')$

The ow-advantage of I is

$$\mathbf{Adv}^{\text{ow}}_{\mathcal{K}_{\text{rsa}}}(I) = \Pr\left[\mathrm{OW}^{I}_{\mathcal{K}_{\text{rsa}}} \Rightarrow \text{true}\right]$$

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \equiv y \pmod{N}$

$\uparrow$ EASY — because $f^{-1}(y) = y^d \bmod N$

Know d

$\uparrow$ EASY — because $d = e^{-1} \bmod \varphi(N)$

Know $\varphi(N)$

$\uparrow$ EASY — because $\varphi(N) = (p-1)(q-1)$

Know p, q

$\uparrow$?

Know N

Factoring problem

Given: N where $N = pq$ and p, q are prime

Find: p, q

If we can factor we can invert RSA. We do not know whether the converse is true, meaning whether or not one can invert RSA without factoring.

A factoring algorithm

Alg $\underline{\text{FACTOR}(N)}$ // $N = pq$ where p, q are primes

for $i = 2, \ldots, \left\lceil \sqrt{N} \right\rceil$ do
  if $N \bmod i = 0$ then
    $p \leftarrow i\,;\ q \leftarrow N/i\,;$ return p, q

This algorithm works but takes time

$$\mathcal{O}(\sqrt{N}) = \mathcal{O}(e^{0.5 \ln N})$$

which is prohibitive.

e.g.: 2048-bit N

Factoring algorithms

Algorithm	Time taken to factor N
Naive	$O(e^{0.5 \ln N})$
Quadratic Sieve (QS)	$O(e^{c(\ln N)^{1/2}(\ln \ln N)^{1/2}})$
Number Field Sieve (NFS)	$O(e^{1.92(\ln N)^{1/3}(\ln \ln N)^{2/3}})$

$\Rightarrow$ need 2048-bit modulus for 128 bit sec. (roughly)

ECM: "smooth" numbers small prime factor

Key size

Current wisdom: For 80-bit security, use a 1024 bit RSA modulus

80-bit security: Factoring takes 2^{80} time.

Factorization of RSA-1024 seems out of reach at present.

Estimates vary, and for more security, longer moduli are recommended.

RSA Cheat Sheet

The RSA function $f(x) = x^e \mod N$ is a trapdoor one way permutation:

- Easy forward: given N, e, x it is easy to compute $f(x)$
- Easy back with trapdoor: Given N, d and $y = f(x)$ it is easy to compute $x = f^{-1}(y) = y^d \mod N$
- Hard back without trapdoor: Given N, e and $y = f(x)$ it is hard to compute $x = f^{-1}(y)$

RSA: one-way   want IND-CPA

much stronger

HOW ???

Plain RSA Encryption

The plain RSA PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ associated to RSA generator $\mathcal{K}_{\text{rsa}}$ is

Alg $\mathcal{K}$	**Alg** $\mathcal{E}_{pk}(M)$	**Alg** $\mathcal{D}_{sk}(C)$
$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$	$C \leftarrow M^e \mod N$	$M \leftarrow C^d \mod N$
$pk \leftarrow (N, e)$; $sk \leftarrow (N, d)$	return C	return M
return (pk, sk)		

Decryption correctness: The "easy-backwards with trapdoor" property implies that for all $M \in \mathbf{Z}_N^*$ we have $\mathcal{D}_{sk}(\mathcal{E}_{pk}(M)) = M$.

Note: The message space is $\mathbf{Z}_N^*$. Messages are assumed to be all encoded as strings of the same length, for example length 4 if $N = 15$.

Attacks? (1) usual attack on det enc.

(2) encrypt yourself and check if ctxt match

"Simple RSA" (SRSA)

The SRSA PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ associated to RSA generator $\mathcal{K}_{\mathrm{rsa}}$ and (public) hash function $H\colon\ \{0,1\}^* \to \{0,1\}^k$ encrypts k-bit messages via:

Alg $\mathcal{K}$	**Alg $\mathcal{E}_{N,e}(M)$**	**Alg $\mathcal{D}_{N,d}(C_a, C_s)$**
$(N,p,q,e,d) \xleftarrow{\$} \mathcal{K}_{\mathrm{rsa}}$	$x \xleftarrow{\$} \mathbf{Z}_N^*$	$x \leftarrow C_a^d \bmod N$
$pk \leftarrow (N,e)$	$K \leftarrow H(x)$	$K \leftarrow H(x)$
$sk \leftarrow (N,d)$	$C_a \leftarrow x^e \bmod N$	$M \leftarrow C_s \oplus K$
return (pk, sk)	$C_s \leftarrow K \oplus M$	return M
	return (C_a, C_s)	

could use AE

fixes the problems by
(1) randomizing enc.
(2) using a hash function.

IND-CPA in RO model.

(1) SHINKS the ciphertext to length of modulus.

(II) uses Feistel rounds

OAEP

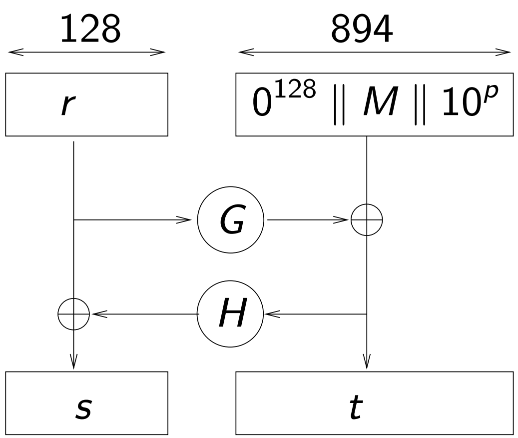
(3) IND-CPA in RO (or std model?)

Receiver keys: $pk = (N, e)$ and $sk = (N, d)$ where $|N| = 1024$

Hash functions: $G\colon \{0,1\}^{128} \to \{0,1\}^{894}$ and $H\colon \{0,1\}^{894} \to \{0,1\}^{128}$

Algorithm $\mathcal{E}_{N,e}(M)$ $\quad // \ |M| \le 765$

$r \xleftarrow{\$} \{0,1\}^{128}$; $p \leftarrow 765 - |M|$



$x \leftarrow s||t$

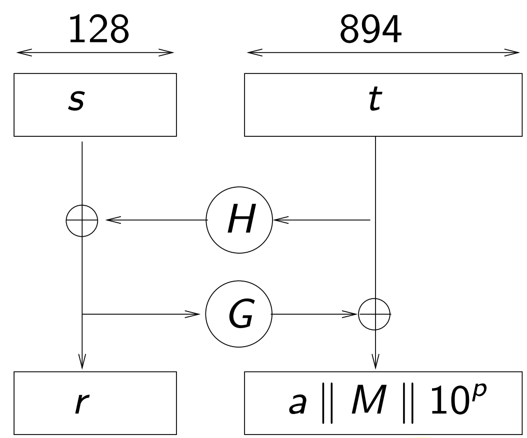
$C \leftarrow x^e \bmod N$

return C

Algorithm $\mathcal{D}_{N,d}(C)$ $\quad // \ C \in \mathbb{Z}_N^*$

$x \leftarrow C^d \bmod N$

$s||t \leftarrow x$



if $a = 0^{128}$ **then return** M

else return $\perp$

KNOW advantages / disavantages btw. RSA-based and DL-based enc. !

OAEP Usage

Protocols:

- SSL ver. 2.0, 3.0 / TLS ver. 1.0, 1.1
- SSH ver 1.0, 2.0
- ...

Standards:

- RSA PKCS #1 versions 1.5, 2.0
- IEEE P1363
- NESSIE (Europe)
- CRYPTREC (Japan)
- ...

Security Results

Scheme	**IND-CPA**?
DHIES	Yes
Plain RSA	No
SRSA	Yes
RSA OAEP	Yes