RSA and ASYMMETRIC (PUBLIC-KEY) ENCRYPTION

RSA Math

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

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

$$(x^e)^d \bmod N = x\ .$$

Proof:

$$\begin{aligned}(x^e)^d \bmod N &= x^{ed \bmod \varphi(N)} \bmod N \\ &= x^1 \bmod N = x\end{aligned}$$

The RSA function

A modulus N and encryption exponent $e \in \mathbf{Z}^*_{\varphi(N)}$ define the RSA function $f : \mathbf{Z}^*_N \to \mathbf{Z}^*_N$ via:

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

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

A value $d \in Z^*_{\varphi(N)}$ satisfying $ed \bmod \varphi(N) = 1$ 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 $x \in \mathbf{Z}^*_N$, the prior claim says that we have

$$f^{-1}(f(x)) = (x^e)^d \bmod N = x\ .$$

Example

Let $N = 15$. So

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

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

Trapdoor permutation

RSA is a trapdoor, one-way permutation:

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

The second is true, to best of our current knowledge, for appropriately-chosen parameters N, e, d.

The choice of parameters is done by an algorithm called an RSA generator.

RSA generators

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

- 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
- $ed \bmod \varphi(N) = 1$

A formula for Phi

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

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

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.

Recall

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

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

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

We use these facts to build RSA generators.

Building RSA generators

Say we wish to have $e = 3$. (We will see that the smaller is e, the more efficient is encryption.) The generator $\mathcal{K}^3_{rsa}$ with (even) security parameter k is as follows:

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

One-wayness of RSA

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

Game $\mathrm{OW}_{\mathcal{K}_{\mathrm{rsa}}}$

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

procedure Finalize(x')
return $(x = x')$

The ow-advantage of I is

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

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$

because $x = y^d \bmod N$

Know d

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$

EASY

because $x = y^d \bmod N$

Know d

EASY

because $d = \text{MOD-INV}(e, \varphi(N))$

Know $\varphi(N)$

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$

EASY

because $x = y^d \bmod N$

Know d

EASY

because $d = \text{MOD-INV}(e, \varphi(N))$

Know $\varphi(N)$

EASY

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

Know p, q

Inverting RSA

Inverting RSA : given N, e, y find x such that $x^e \bmod N = y$

EASY

because $x = y^d \bmod N$

Know d

EASY

because $d = \text{MOD-INV}(e, \varphi(N))$

Know $\varphi(N)$

EASY

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

Know p, q

?

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 $i = 2, \ldots, \lceil \sqrt{N} \rceil$ 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.

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)	$O(e^{1.92(\ln N)^{1/3}(\ln \ln N)^{2/3}})$

Factoring records

bit-length of number	When factored	Algorithm used
400	1993	QS
428	1994	QS
431	1996	NFS
465	1999	NFS
515	1999	NFS
576	2003	NFS
768	2009	NFS
795	2019	NFS
829	2020	NFS

Moduli sizes

We estimate that a 1024-bit RSA modulus provides 80 bits of security, meaning factoring it takes 2^{80} time.

Factorization of a 1024-bit modulus hasn't been done yet in public, but is within reach of large organizations. Longer moduli, like 2048 bits, have been recommended since around 2010.

Just because factoring some large numbers seems to be hard does not mean factoring *all* large numbers is hard.

For example, a random integer has probability $1/2$ of having 2 as a prime factor.

This is why RSA uses moduli N designed to resist known factoring algorithms.

Choices of encryption exponent

Common choices are $e = 3$, $e = 17$ and $e = 65,537$. Why these?

e	bin(e)
3	11
17	10001
65,537	1000000000000001

Recall that the modular exponentiation algorithm computing $x \mapsto x^e \bmod N$ uses $c(b)$ modular multiplications per bit $b \in \{0, 1\}$ in the binary expansion bin(e), where $c(0) = 1$ and $c(1) = 2$. So the fewer the number of 1s in bin(e), the faster is the operation.

Textbook RSA is insecure

Choosing a good modulus and exponent is not enough to make an RSA implementation secure.

For example:

- Let $e = 3$, N have 1024 bits, and encrypt $y = x^e \bmod N$.

To avoid this, typical RSA implementations *pad* messages before encrypting.

But even this is hard to get right: the most common RSA padding scheme in use is broken if it throws an error when it discovers incorrect padding.

RSA Video

http://www.youtube.com/watch?v=wXB-V_Keiu8

RSA: what to remember

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 $x = f^{-1}(y)$

The quantum threat

On a quantum computer, Shor's algorithm can compute discrete logarithms and factor in polynomial time.

Efforts to build quantum computers are underway.

Efforts are underway to standardize public-key cryptography based on computational problems like finding short vectors in lattices for which there are currently no known efficient quantum algorithms.

Two settings

Symmetric encryption:

- Before Alice and Bob can communicate securely, they need to have a common secret key K_{AB}.
- If Alice wishes to also communicate with Charlie then she and Charlie must also have another common secret key K_{AC}.
- If Alice generates K_{AB}, K_{AC}, they must be communicated to her partners over private and authenticated channels.

Asymmetric (public-key) encryption:

- Alice has a secret decryption key dk that is shared with nobody, and an associated public encryption key ek that is known to everybody.
- Anyone (Bob, Charlie, ...) can use Alice's encryption key ek to send her an encrypted message which only she can decrypt.

Syntax of a PKE scheme

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

- $(ek, dk) \xleftarrow{\$} \mathcal{K}$ — generate an encryption key ek and matching decryption key dk
- $C \xleftarrow{\$} \mathcal{E}_{ek}(M)$ — encrypt message M under encryption key ek to get a ciphertext C. Algorithm $\mathcal{E}$ may be randomized.
- $M' \leftarrow \mathcal{D}_{dk}(C)$ — decrypt ciphertext C under decryption key dk to get an output $M' \in \{0,1\}^* \cup \{\perp\}$.

Correct decryption requirement

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

$$\Pr[\mathcal{D}_{dk}(\mathcal{E}_{ek}(M)) = M] = 1$$

for all (ek, dk) 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. It may depend on the encryption key.

How it works

Step 1: Key generation
Alice locally computes $(ek, dk) \xleftarrow{\$} \mathcal{K}$ and stores dk.

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

Step 3: The sender encrypts a message M under ek and sends the ciphertext C to Alice.

Step 4: Alice decrypts C under dk to recover M.

Verifying the authenticity of public keys

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

This is solved in a variety of imperfect ways in practice. One could:

- Use certificates as we will see later. (TLS)
- Verify a fingerprint/hash of the public key through an out of band channel. (Signal)
- Use existing trusted infrastructure, like social media accounts. (Keybase)
- Build a social network of digital signatures. (PGP)
- Trust on first use and verify fingerprint/hash in the future. (SSH)

Security of PKE Schemes

We formalize two goals:

- **IND-CPA**: Indistinguishability under chosen-plaintext attack. Just like for symmetric encryption, except that adversary needs to be given the encryption key.
- **IND-CCA**: Indistinguishability under chosen-ciphertext attack. A stronger goal in which the adversary has (limited) access to a decryption oracle.

PKE IND-CPA security games

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

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

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

$$\Pr\left[\text{Left}^A_{\mathcal{AE}} \Rightarrow 1\right] \qquad \Pr\left[\text{Right}^A_{\mathcal{AE}} \Rightarrow 1\right]$$

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

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

PKE IND-CPA: Explanations

The "return ek" statement in **Initialize** means the adversary A gets the encryption key ek as input. It does not get dk.

It can call **LR** with any equal-length messages M_0, M_1 of its choice to get back an encryption $C \xleftarrow{\$} \mathcal{E}_{ek}(M_b)$ of M_b under ek, 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.

Hybrid encryption

Given:

- A KEM $\mathcal{KE} = (\mathcal{KK}, \mathcal{EK}, \mathcal{DK})$ with key length k
- A symmetric encryption scheme $\mathcal{SE} = (\mathcal{KS}, \mathcal{ES}, \mathcal{DS})$ for which $\mathcal{KS}$ returns random k-bit keys.

Hybrid encryption associates to the above a PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ via:

Alg $\mathcal{K}$	**Alg** $\mathcal{E}_{ek}(M)$	**Alg** $\mathcal{D}_{dk}((C_a, C_s))$
$(ek, dk) \xleftarrow{\$} \mathcal{KK}$	$(K, C_a) \xleftarrow{\$} \mathcal{EK}_{ek}$	$K \leftarrow \mathcal{DK}_{dk}(C_a)$
Return (ek, dk)	$C_s \xleftarrow{\$} \mathcal{ES}_K(M)$	$M \leftarrow \mathcal{DS}_K(C_s)$
	Return (C_a, C_s)	Return M

Above, it is understood that if any input to an algorithm is $\perp$, then so is the output.

Hybrid encryption works

If the KEM and symmetric encryption scheme are both IND-ATK, then so is the PKE scheme constructed by hybrid encryption.

Theorem: Let $\mathcal{KE} = (\mathcal{KK}, \mathcal{EK}, \mathcal{DK})$ be a KEM with key length k. Let $\mathcal{SE} = (\mathcal{KS}, \mathcal{ES}, \mathcal{DS})$ be a symmetric encryption scheme for which $\mathcal{KS}$ returns random k-bit keys. Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the corresponding PKE scheme built via hybrid encryption. Let $\mathrm{atk} \in \{\mathrm{cpa}, \mathrm{cca}\}$. Let A be an adversary making q_e **LR** queries. Then there are adversaries B_a, B_s such that

$$\mathbf{Adv}^{\text{ind-atk}}_{\mathcal{AE}}(A) \leq 2 \cdot \mathbf{Adv}^{\text{ind-atk}}_{\mathcal{KE}}(B_a) + q_e \cdot \mathbf{Adv}^{\text{ind-atk}}_{\mathcal{SE}}(B_s) \ .$$

The number of **Enc** queries of B_a is q_e. The number of **LR** queries of B_s is 1. In the $\mathrm{atk} = \mathrm{cca}$ case, B_a, B_s each make the same number of **Dec** queries as A. The running times of B_a, B_s are about the same as that of A.

Benefits of hybrid encryption

Modular design: Many choices of components, KEMs are simpler than PKE schemes.

Assurance via proof as per above Theorem saying hybrid encryption works.

Speed: The block ciphers and hash functions used in symmetric cryptography are much faster (factors of 100x to 10,000x depending on platforms) than the operations on numbers used for asymmetric cryptography.

So performance is improved by limiting the number-theoretic operations as in hybrid encryption.

Where we are

We know how to achieve IND-ATK-secure PKE given

- An IND-ATK-secure KEM, and
- An IND-ATK-secure symmetric encryption scheme

We have plenty of symmetric encryption schemes:

- For the IND-CPA case: AES-CTR$, AES-CBC$, ...
- For the IND-CCA case: Encrypt-then-Mac, OCB, GCM, ...

But simpler, deterministic choices are possible too, since security is only required against adversaries B_s making 1 **LR** query.

We need KEMs.

We will build KEMs using number theory, considering in turn using the DL problem and using RSA.

Hashing in KEMs

Our KEMs may use (public, keyless) functions $\mathbf{H}_i$: $\{0,1\}^* \to \{0,1\}^{\ell_i}$, for $1 \leq i \leq n$.

The number n of them, and their output lengths, depend on the scheme. Usually $n = 1$ or $n = 2$.

In practice (implementation), these are built from cryptographic hash functions as discussed next.

Proofs of security for the KEMs use the Random Oracle Model (ROM) [BR93] in which $\mathbf{H}_1, \ldots, \mathbf{H}_n$ are modeled as independent random functions. They are formalized as game procedures to which scheme algorithms, *as well as the adversary*, have oracle access, and are thus called Random Oracles (ROs).

Practical choices for the $\mathbf{H}_i$

We seek suitable functions $\mathbf{H}_i$: $\{0,1\}^* \to \{0,1\}^{\ell_i}$, for $1 \leq i \leq n$.

$\mathrm{SHAKE256}$ is an XOF (eXtendable Output length Function) that takes an input indicating the number of output bits returned.

So we could set $\mathbf{H}_i(x) = \mathrm{SHAKE256}(\langle i \rangle \| x, \ell_i)$ where $\langle i \rangle$ is a 1-byte encoding of i and we assume $n < 2^8$.

We could also set $\mathbf{H}_i(x)$ to the first ℓ_i bits of the sequence

$$\mathrm{SHA256}(\langle 0 \rangle \| \langle i \rangle \| x) \,\|\, \mathrm{SHA256}(\langle 1 \rangle \| \langle i \rangle \| x) \,\|\, \cdots \,\|\, \mathrm{SHA256}(\langle 2^8 - 1 \rangle \| \langle i \rangle \| x)$$

This assumes $\ell_i \leq 2^8 \cdot 256$.

Heuristically, we desire that $\mathbf{H}_1, \ldots, \mathbf{H}_n$ "behave like independent random functions." But there is no corresponding formal definition.

KEMs from DL?

Let $G = \langle g \rangle$ be a cyclic group of order m in which the DL problem is hard. Can we design a KEM $\mathcal{KE} = (\mathcal{KK}, \mathcal{EK}, \mathcal{DK})$ whose IND-CPA security reduces to DL?

How about: Let the receiver's encryption key be g. Let $\mathcal{EK}_g$ pick $x \xleftarrow{\$} \mathbf{Z}_m$ and return (x, X) where $X = g^x$.

Then obtaining x from X requires solving DL, and would be hard for an adversary.

So are we done?

KEMs from DL?

Let $G = \langle g \rangle$ be a cyclic group of order m in which the DL problem is hard. Can we design a KEM $\mathcal{KE} = (\mathcal{KK}, \mathcal{EK}, \mathcal{DK})$ whose IND-CPA security reduces to DL?

How about: Let the receiver's encryption key be g. Let $\mathcal{EK}_g$ pick $x \xleftarrow{\$} \mathbf{Z}_m$ and return (x, X) where $X = g^x$.

Then obtaining x from X requires solving DL, and would be hard for an adversary.

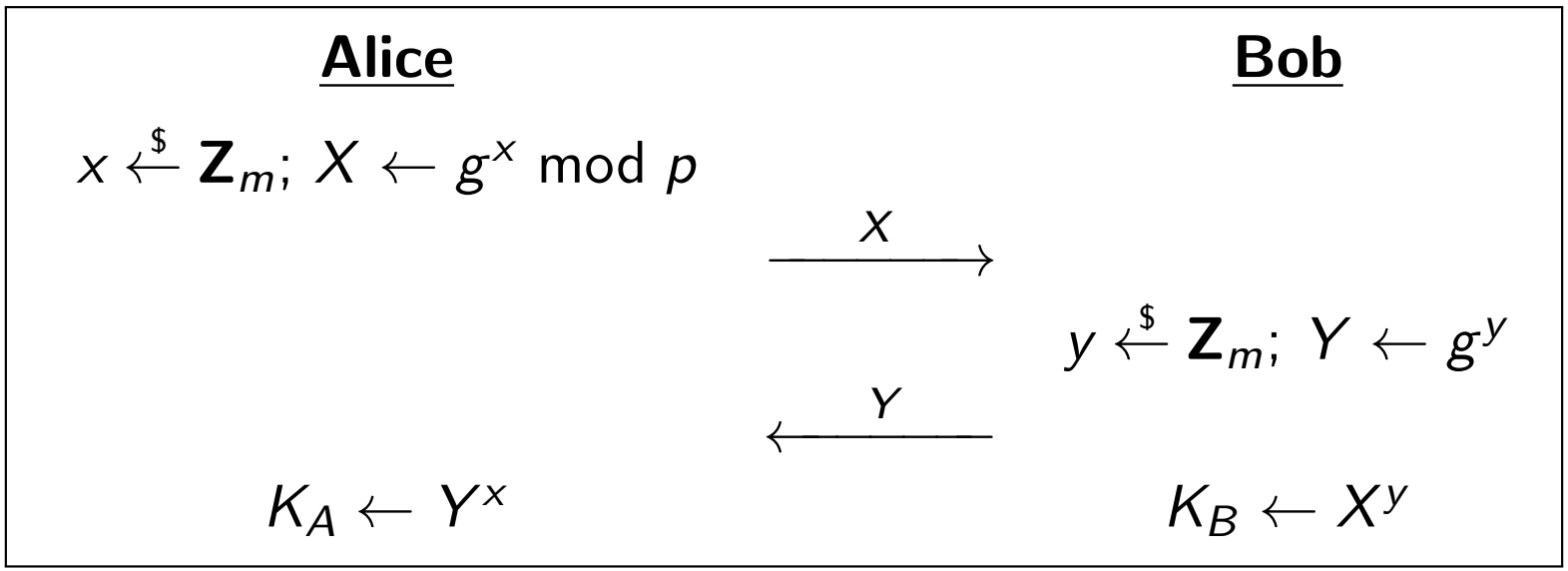
So are we done?

No. The legitimate receiver has no way to decrypt X, to obtain x, short of computing DL.

A sign that something is amiss is that, in the above scheme, the receiver has no decryption key.

Recall DH Secret Key Exchange

Let $G = \langle g \rangle$ be a cyclic group of order m.



- $Y^x = (g^y)^x = g^{xy} = (g^x)^y = X^y$, so $K_A = K_B$
- Adversary is faced with the CDH problem, which needs to be assumed hard for security. This is a stronger requirement than hardness of DL.

From key exchange to PKE

We can turn DH key exchange into a PKE scheme via

- Alice has encryption key $X = g^x$ and decryption key $x \xleftarrow{\$} \mathbf{Z}_{p-1}$
- If Bob wants to encrypt message M for Alice, he
  - Picks $y \xleftarrow{\$} \mathbf{Z}_{p-1}$ and sends $Y = g^y$ to Alice
  - Computes $Z = (g^x)^y = g^{xy}$, hashes it to get a key K, encrypts M symmetrically under K to get a ciphertext C_s, and sends C_s to Alice.
- Alice can recompute $Z = Y^x = g^{xy}$ using her decryption key x. Then she can recompute K and decrypt C_s under it to get M.

The adversary is faced with either solving CDH or breaking the symmetric encryption scheme.

The DHIES scheme

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

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)	$W \leftarrow H(K) \oplus M$	return M
	return (Y, W)	

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

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

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

ECIES is DHIES with the group being an elliptic curve group.

ECIES features:

Operation	**Cost**
encryption	2 160-bit exp
decryption	1 160-bit exp
ciphertext expansion	160 bits

ciphertext expansion = (length of ciphertext) - (length of plaintext)

Plain-RSA PKE scheme

Let $\mathcal{K}_{\text{rsa}}$ be an RSA generator. The plain RSA PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined via:

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

Above, (N, e) is the encryption key and (N, d) is the decryption key.

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

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 $N = 15$.

Security of the Plain-RSA PKE scheme

Let $\mathcal{K}_{\text{rsa}}$ be an RSA generator. The plain RSA PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is defined via:

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

Getting d from (N, e) involves factoring N.

But $\mathcal{E}$ is deterministic so we can detect repeats and the scheme is not IND-CPA secure.

The SRSA scheme

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

Alg $\mathcal{K}$

$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$
$ek \leftarrow (N, e)$
$dk \leftarrow (N, d)$
return (ek, dk)

Alg $\mathcal{E}_{N,e}(M)$

$x \xleftarrow{\$} \mathbf{Z}_N^*$
$K \leftarrow H(x)$
$C_a \leftarrow x^e \bmod N$
$C_s \leftarrow K \oplus M$
return (C_a, C_s)

Alg $\mathcal{D}_{N,d}(C_a, C_s)$

$x \leftarrow C_a^d \bmod N$
$K \leftarrow H(x)$
$M \leftarrow C_s \oplus K$
return M

Security of 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: $\{0,1\}^* \to \{0,1\}^k$ can be proven IND-CPA assuming

- $\mathcal{K}_{\mathrm{rsa}}$ is one-way
- H is a "random oracle," meaning a "perfect" hash function.

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

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

PKE summary

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