CSE 107: Introduction to Modern Cryptography

Nadia Heninger

UCSD

Winter 2025 Lecture 14

Last time:

- Baby step giant step discrete log
- RSA, public-key encryption

This time:

- Public-key cryptography: RSA, factoring assumptions, KEMs, hybrid encryption

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.

Chosen-ciphertext attacks

Left-Right CCA security definition for symmetric encryption:

Game $\mathrm{Left}_{\mathcal{SE}}$	Game $\mathrm{Right}_{\mathcal{SE}}$
procedure Initialize $K \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$	**procedure** Initialize $K \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$
procedure LR(M_0, M_1) $C \xleftarrow{\$} \mathcal{E}_K(M_0)$; $S \leftarrow S \cup \{C\}$ Return C	**procedure LR**(M_0, M_1) $C \xleftarrow{\$} \mathcal{E}_K(M_1)$; $S \leftarrow S \cup \{C\}$ Return C
procedure Dec(C) If $C \in S$ then return $\perp$ return $M \leftarrow \mathcal{D}_K(C)$	**procedure** Dec(C) If $C \in S$ then return $\perp$ return $M \leftarrow \mathcal{D}_K(C)$

The (ind-cca) advantage of A is

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

Chosen-ciphertext attack security for asymmetric encryption

Left-Right CCA security definition for asymmetric encryption:
Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a PKE scheme and A an adversary.

Game $\mathrm{Left}_{\mathcal{AE}}$

procedure Initialize
$(ek, dk) \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$;
return ek

procedure LR(M_0, M_1)
$C \xleftarrow{\$} \mathcal{E}_{ek}(M_0)$; $S \leftarrow S \cup \{C\}$
Return C

procedure Dec(C)
If $C \in S$ then return $\perp$
return $M \leftarrow \mathcal{D}_{dk}(C)$

Game $\mathrm{Right}_{\mathcal{AE}}$

procedure Initialize
$(ek, dk) \xleftarrow{\$} \mathcal{K}$; $S \leftarrow \emptyset$;
return ek

procedure LR(M_0, M_1)
$C \xleftarrow{\$} \mathcal{E}_{ek}(M_1)$; $S \leftarrow S \cup \{C\}$
return C

procedure Dec(C)
If $C \in S$ then return $\perp$
return $M \leftarrow \mathcal{D}_{dk}(C)$

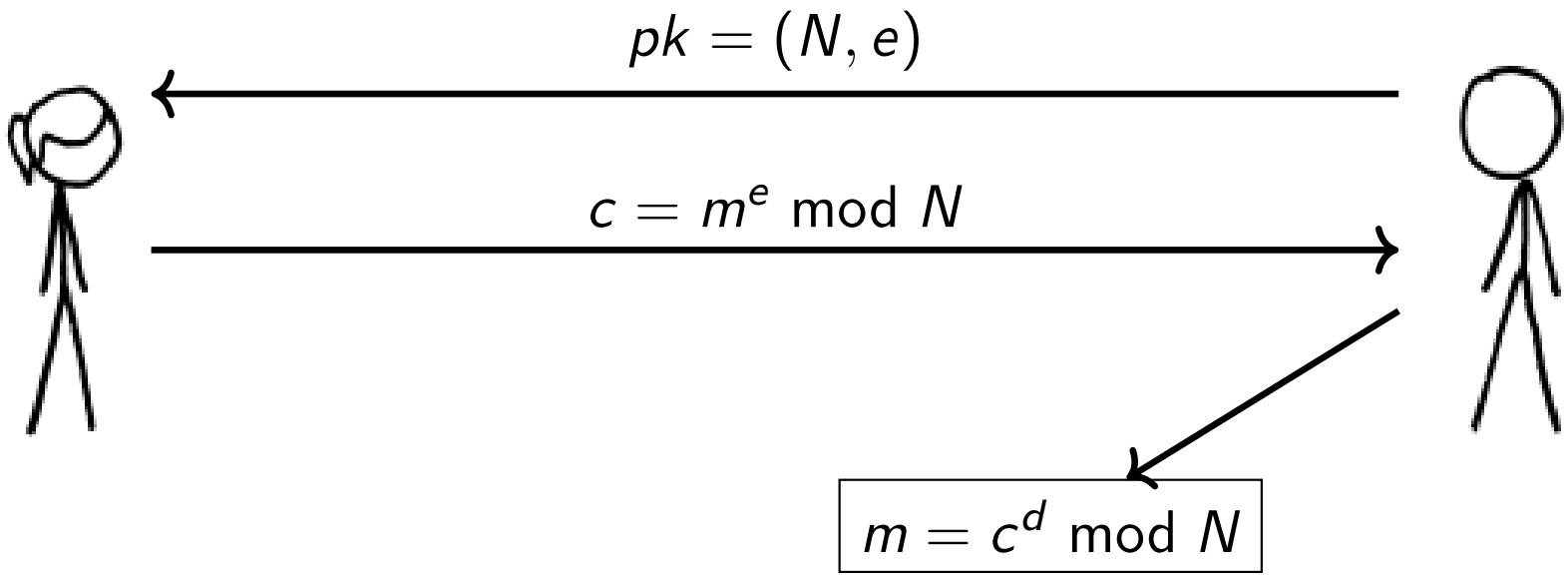
The (ind-cca) advantage of A is

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

Reminder: Textbook RSA Encryption

[Rivest Shamir Adleman 1977]

- Key Generation:
  1. Generate random primes p, q
  2. $N = pq$
  3. Choose odd e s.t. $\gcd(e, (p-1)(q-1)) = 1$
  4. $d = e^{-1} \bmod (p-1)(q-1)$
  5. $pk = (N, e)$, $sk = (N, d)$.
- Encryption: $c = m^e \bmod N$
- Decryption: $m = c^d \bmod N$



RSA is homomorphic under multiplication

If we have a ciphertext $c = m^e \bmod N$, can forge encryption of mr by computing

$$cr^e \bmod N = m^e r^e \bmod N = (mr)^e \bmod N$$

Implications:

- Positive use: blinding. Can blind ciphertexts before decryption to try to prevent side-channel attacks, or blind signatures before signing. (More later.)
- Negative use: Chosen ciphertext attacks.

Textbook RSA is not CCA-secure

Textbook RSA as we described it is not CCA-secure: there is no ciphertext integrity.

Textbook RSA is not CCA-secure

Textbook RSA as we described it is not CCA-secure: there is no ciphertext integrity.

1. Input challenge ciphertext $c = m^e \bmod N$.
2. Submit ciphertext $c' = r^e c \bmod N$ for decryption.
3. Receive message $m' = rm$.
4. Original message is $m' r^{-1} \bmod N = m$.

Textbook RSA is insecure

Some variants of RSA have even easier attacks.

Small e attack:

If $e = 3$ and $m < N^{1/3}$, $m = c^{1/3}$ over $\mathbb{Z}$.

Textbook RSA is insecure

Some variants of RSA have even easier attacks.
Small e attack:
If $e = 3$ and $m < N^{1/3}$, $m = c^{1/3}$ over $\mathbb{Z}$.

Cube roots over $\mathbb{Z}$: polynomial time

Cube roots over $\mathbb{Z}/N\mathbb{Z}$: not efficient unless factorization of N is known

Textbook RSA is insecure

Meet-in-the-middle attack for random m

Input ciphertext c.

- Compute $x_i = c/r^e \bmod N$ for r from $1, \ldots, \sqrt{m}$.
- Compute $y_i = s^e \bmod N$ for s from $1, \ldots, \sqrt{m}$.
- If $y_i = x_j$ return $r \cdot s$.

If $m = r \cdot s$, $m^e = r^e \cdot s^e$.

For a randomly chosen m that isn't too large, good probability it has no large factors.

Re-frame public-key encryption: We only ever want to use it to encrypt symmetric key material.

We can then use a symmetric authenticated encryption algorithm to encrypt our actual data.

Symmetric encryption is *much much* faster than slow public key mathematical operations.

We will rename public key encryption algorithms to be Key Encapsulation Mechanism (KEM).

Hybrid encryption, formally

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.

Using RSA with hybrid encryption, concretely

Use a symmetric cipher $(\mathsf{SymEnc}, \mathsf{SymDec})$ and a hash function H.

- Key Generation:
  1. Generate primes p, q; $N = pq$
  2. Choose odd e s.t. $\gcd(e, \phi(N)) = 1$
  3. $d = e^{-1} \bmod \phi(N)$
  4. $pk = (N, e)$, $sk = (N, d)$.

- Encryption: Choose random x, $y = x^e \bmod N$; $k = H(x)$;
  $$c = \mathsf{SymEnc}_k(m), \text{ send } (y, c)$$

- Decryption: Input (y, c), compute $x = y^d \bmod N$; $k = H(x)$;
  $m = \mathsf{SymDec}_k(c)$

Unfortunately, nobody actually does this in practice.

RSA Padding Schemes

To protect against RSA malleability, RSA is universally used with a padding scheme in practice.

Instead of $\text{Enc}_{pk}(m) = m^e \bmod N$, we define:

- $\text{Enc}_{pk}(m) = (\text{pad}(m))^e \bmod N$
- $\text{Dec}_{sk}(c)$:
  1. Compute $p = c^d \bmod N$.
  2. If p has correct padding format, return $\text{unpad}(p)$.
  3. Else return "failure".

PKCS #1 v. 1.5 padding

PKCS #1 v. 1.5 padding is the most common padding scheme for RSA in practice.
Encryption:

```
m = 00 02 [random padding string] 00 [data]
```

Signatures:

```
m = 00 01 FF .. FF 00 [data]
```

To decrypt, implementation checks padding format:

- First two byes correct.
- Padding string contains no null bytes.
- Presence of null byte.
- `data` is typically symmetric key data.

Decades of RSA padding vulnerabilities

Unfortunately, the padding schemes on the previous slide are:

- Not CCA-secure
- Prone to implementation shortcuts
- Impossible to move away from because of backwards compatibility issues.

This has resulted in decades of problems for RSA as an encryption scheme.

At this point, there is essentially no reason ever to use RSA encryption. We're teaching it to you because it's easy to understand, but please use something better.

The factoring problem

Input: N where $N = pq$ and p, q are primes.

Output: 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.

Trial division factoring algorithm

for $i = 2, \ldots, \sqrt{N}$:
    if $N \bmod i \equiv 0$:
        return i, N/i

This algorithm takes time $O(\sqrt{N})$ which is exponential in $\lg N$.

Trial division factoring algorithm

for $i = 2, \ldots, \sqrt{N}$:
    if $N \bmod i \equiv 0$:
        return i, N/i

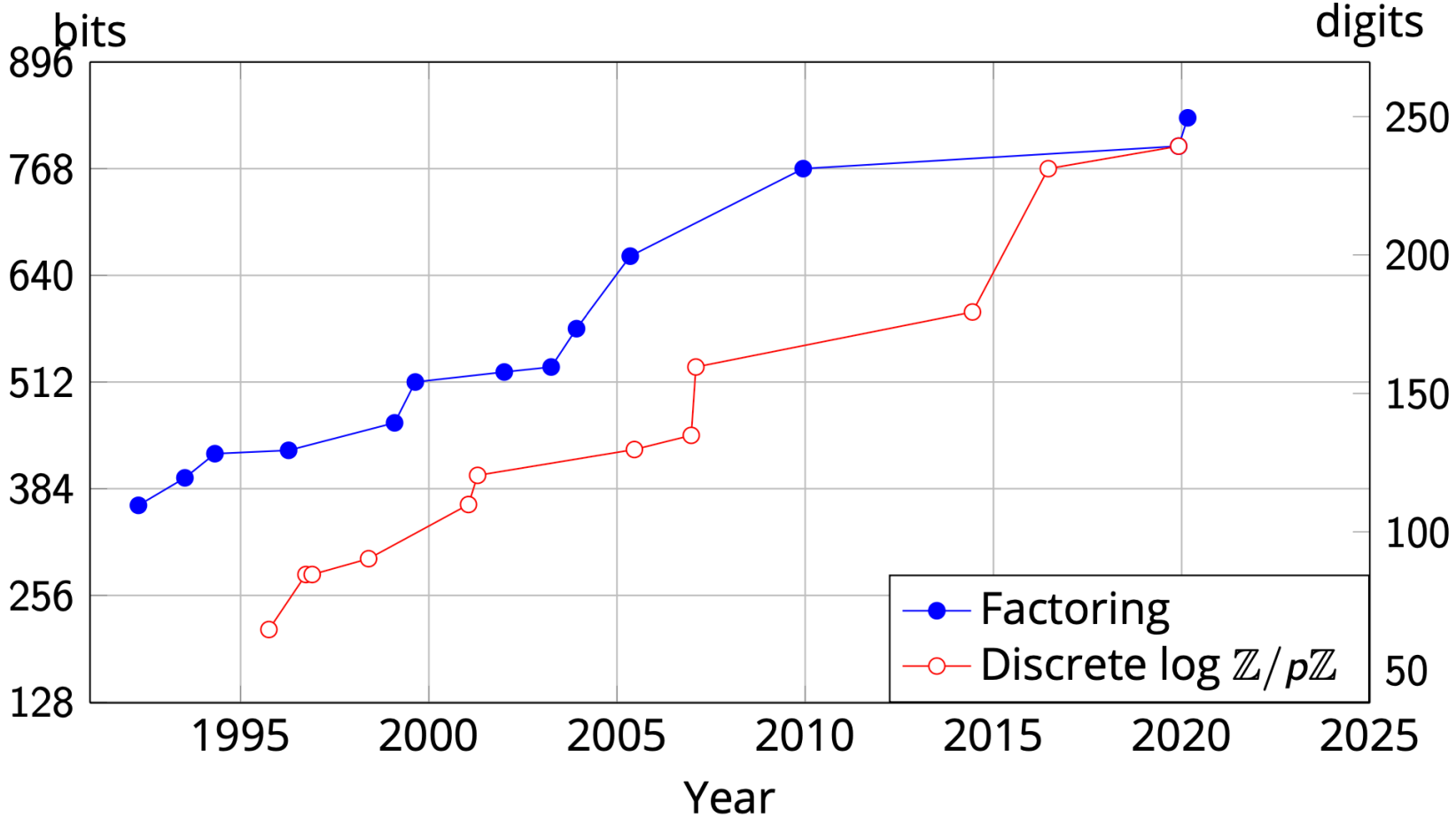
This algorithm takes time $O(\sqrt{N})$ which is exponential in $\lg N$.

However, this algorithm makes it clear that factoring is not hard for *all* integers.

Factoring algorithms

Algorithm	**Running time**
Trial Division	$O(e^{0.5 \ln N})$
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 and discrete logarithm records



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.

Public-key encryption from discrete log: ElGamal

Global parameters: A cyclic group G with generator g of prime order q

- Key Generation: Choose $a \in \mathbb{Z}_q$ and compute $u = g^a$. Then $pk = u$ and $sk = a$.
- Encryption: Input public key $pk = u$. Choose $b \in \mathbb{Z}_q$. Compute $v = g^b$. $\mathrm{Enc}_{pk}(m) = (v, u^b \cdot m)$.
- Decryption: Input a ciphertext (v, e), secret key $sk = a$. $\mathrm{Dec}_{sk}((v, e)) = e/v^a$.

This can be proven to be secure, but easy to implement poorly in practice and basically nobody uses it.

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.

We have initial standards for public-key cryptography based on computational problems like finding short vectors in lattices for which there are currently no known efficient quantum algorithms.