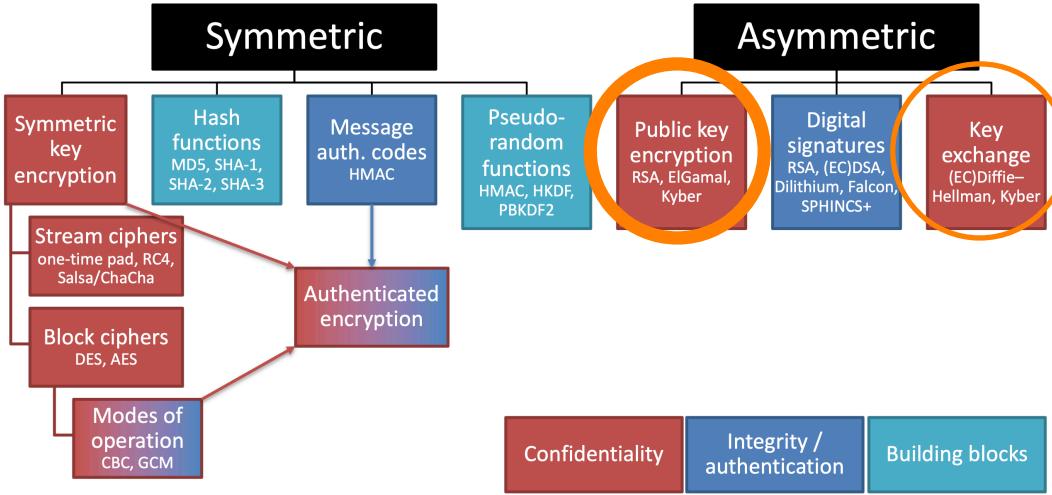
Topic 3.4
Public key cryptography – Security of public key encryption

Douglas Stebila

CO 487/687: Applied Cryptography
Fall 2024

Map of cryptographic primitives



Security of RSA encryption

Is RSA encryption secure?

What does it mean to be secure?

Security of RSA is related to factoring. Is factoring hard?

Outline

Security definitions for public key encryption

Security of RSA encryption

Difficulty of factoring

Security of Diffie–Hellman key exchange and Elgamal encryption

Definition of public-key encryption

Definition (Public key encryption scheme)

A public-key encryption scheme consists of:

- M – the plaintext space; C – the ciphertext space,
- K_{pubkey} – the space of public keys; K_{privkey} – the space of private keys,
- A randomized algorithm $\mathcal{G} \to K_{\text{pubkey}} \times K_{\text{privkey}}$, called a key-generation algorithm,
- A (possibly randomized) encryption algorithm $\mathcal{E}\colon K_{\text{pubkey}} \times M \to C$,
- A decryption algorithm $\mathcal{D}\colon K_{\text{privkey}} \times C \to M$.

Definition (Correctness of public key encryption)

For any key pair $(k_{\text{pubkey}}, k_{\text{privkey}})$ produced by $\mathcal{G}$, and all $m \in M$,

$$\mathcal{D}(k_{\text{privkey}}, \mathcal{E}(k_{\text{pubkey}}, m)) = m.$$

A framework for security definitions

Recall that for a symmetric-key encryption scheme, security depends on three questions:

1. How does the adversary interact with the communicating parties?
2. What are the computational powers of the adversary?
3. What is the adversary's goal?

Basic assumption (Kerckhoffs's principle, Shannon's maxim): The adversary knows everything about the algorithm, except the secret key k. (Avoid security by obscurity!!)

The same principles also apply to public-key cryptography.

(Replace "secret key" by "private key")

1. Adversary's Interaction

Possible methods of attacks against a public-key encryption scheme:

- Passive attacks:
  - Key-only attack: The adversary is given the public key(s).
    We always assume the adversary has the public key(s).
  - Chosen-plaintext attack: The adversary can choose some plaintext(s) and obtain the corresponding ciphertext(s). Equivalent to a key-only attack.
- Active attacks:
  - Non-adaptive chosen-ciphertext attack: The adversary can choose some ciphertext(s) and obtain the corresponding plaintext(s).
  - (Adaptive) chosen-ciphertext attack: Same as above, except the adversary can also iteratively choose which ciphertexts to decrypt, based on the results of previous queries.

3. Adversary's goal

Possible goals when attacking a public-key encryption scheme:

- Total break: Determine the private key, or determine information equivalent to the private key.
- Decrypt a given ciphertext: Adversary is given a ciphertext c and decrypts it (without querying for the decryption of c).
- Learn some partial information about a message: Adversary is given/chooses a ciphertext c and learns some partial information about the decryption of c (without querying for the decryption of c).

System designer's goals

The designer's goals are the opposite of the adversary's goals.

A public-key encryption scheme is defined to be:

- Totally insecure if the adversary can obtain the private key.
- One-way if the adversary cannot decrypt a given ciphertext.
- Semantically secure if the adversary cannot learn any partial information about a message.

As with symmetric encryption, it is often easier to work with the indistinguishability security notion which is equivalent to semantic security

- an adversary who picks two messages m_0, m_1, and then is given the encryption of one of them $c = \mathcal{E}(pk, m_b)$, cannot decide which it was given.

Constructing full security definitions

A full security definition consists of a security goal, a method of attack, and a statement of the computational power of the adversary. For example, here are three different security definitions:

- One-way secure under a chosen-plaintext attack by a computationally bounded adversary (OW-CPA).
- Semantically secure / indistinguishable under a chosen-plaintext attack by a computationally bounded adversary (IND-CPA).
- Semantically secure / indistinguishable under an adaptive chosen-ciphertext attack by a computationally bounded adversary (IND-CCA2).

Outline

Security definitions for public key encryption

Security of RSA encryption

Difficulty of factoring

Security of Diffie–Hellman key exchange and Elgamal encryption

Insecurity of RSA

Theorem

If integer factorization is easy, then RSA encryption is totally insecure.

Proof.

Given (n, e).

Factor n into p and q.

Compute $d = e^{-1} \bmod (p-1)(q-1)$.

Return secret key (n, d). □

Security of RSA

Theorem

If RSA is totally insecure, then integer factorization is easy.

Proof.

Suppose, for any RSA public key (n, e), we can calculate (n, d).

We can factor n using the following algorithm:

1. Repeat say 20 times:
   1.1 Choose (n, e_i) where e_i is a large prime less than n
   1.2 Compute (n, d_i)
   1.3 Let $x_i = e_i d_i - 1$
2. Let $x = \gcd(x_1, x_2, \ldots, x_{20})$. Note that x is very likely equal to $\phi(n) = (p-1)(q-1)$.
3. Let $\{p, q\} = \frac{1+n-x \pm \sqrt{(1+n-x)^2 - 4n}}{2}$

With high probability, p and q will be the prime factors of n. □

The RSA problem

Definition (The RSA problem)

Given

- An RSA public key (n, e), and
- An element $c \in \mathbb{Z}_n$ such that $\gcd(c, n) = 1$,

find an element $m \in \mathbb{Z}_n$ such that

$$c = m^e \bmod n.$$

One-way security

Theorem

If the RSA problem is computationally infeasible, then the RSA public-key encryption scheme is one-way secure under a chosen plaintext attack (OW-CPA).

Proof.

Follows immediately from the definitions of OW-CPA and the RSA problem. □

Semantic security

Theorem

*RSA is **not** semantically secure under a ciphertext-only attack.*

Proof.

Let $c = m^e \bmod n$ be an encryption of a message m under the RSA public key (n, e).

The algorithm below determines some partial information about m:

1. Does $c = 1$?
   - 1.1 Yes — then $m = 1$.
   - 1.2 No — then $m \neq 1$.

Therefore RSA is not semantically secure. □

Summary of RSA security

- The version of RSA presented in the original publication is nowadays called “Textbook RSA.”
- Textbook RSA is provably:
  - Not totally insecure, assuming integer factorization is hard.
  - One-way secure, assuming the RSA problem is hard.
  - Not semantically secure.
- Open problem: Is integer factorization equivalent to the RSA problem?

(Im)possibility of semantic security

A deterministic encryption algorithm (such as RSA) cannot yield semantic security.

- Given a ciphertext c and a public key, choose m at random and compute $c' = E_{\text{pubkey}}(m)$.
- If $c = c'$ then we know the plaintext was m.
- If $c \neq c'$ then we know the plaintext was **not** m.
- Either way, we have learned information about the plaintext.

Achieving semantic security

Strategy #1: Add random padding to RSA.

- Naive padding is insecure under chosen-plaintext attack (Coppersmith's attack)
- PKCS #1 v1.5 padding: not semantically secure under chosen-ciphertext attack (Bleichenbacher)
- RSA-OAEP / PKCS #1 v2.1: provably semantically secure, assuming the RSA problem is hard (Bellare and Rogaway)

Strategy #2: Use randomized encryption, such as in Elgamal encryption.

Security benefits from randomized encryption

Definition (Semantic security of public key encryption)

Recall: An encryption scheme is *semantically secure* if an adversary cannot learn any information about a plaintext from the corresponding ciphertext (except possibly its length).

- A deterministic encryption algorithm (such as RSA) cannot yield semantic security.
  - Given a ciphertext c and a public key, choose m at random and compute $c' = E_{\text{pubkey}}(m)$.
  - If $c = c'$ then we know the plaintext was m.
  - If $c \neq c'$ then we know the plaintext was **not** m.
  - Either way, we have learned information about the plaintext.
- A randomized encryption algorithm avoids this problem.
  - With randomized encryption, even if $c = E_{\text{pubkey}}(m)$ and $c' = E_{\text{pubkey}}(m)$, we have (typically) $c \neq c'$.
- Randomized encryption is a *necessary* but not *sufficient* condition for semantic security.

Randomized encryption algorithms

- RSA does not use a randomized encryption algorithm, but one can modify the encryption algorithm to make it randomized.
  - One approach is to add random padding.
  - Standardized in PKCS#1.

Randomized encryption algorithms

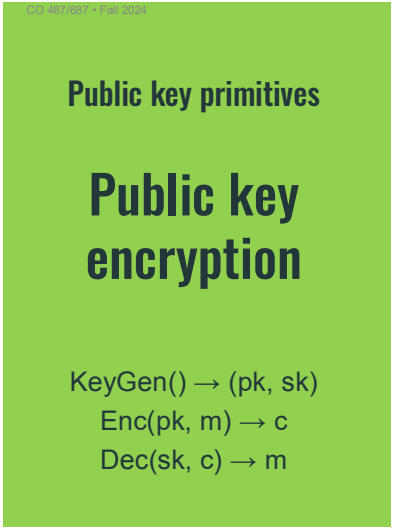
- Elgamal uses a randomized encryption algorithm: $E(m) = (g^r, m \cdot (g^x)^r)$ where r is random.
- (Tsiounis and Yung, 1998) Elgamal is semantically secure under the Decisional Diffie-Hellman assumption if the message space M is the group $\langle g \rangle$ generated by the element g.

Decisional Diffie-Hellman assumption (DDH)

Let $x, y, z \in_R \mathbb{Z}_q$. Given g, g^x, g^y, and either g^{xy} or g^z, it is computationally infeasible to determine whether you were given the real g^{xy} or the random g^z.

- The DDH assumption does not always hold!
  - DDH is known to be false when g is a generator of $\mathbb{Z}_p^*$.
  - DDH also fails in some elliptic curve groups.

4+8+7 things to remember from CO 487



- **Provides confidentiality.**
- Security goal: given public key, semantic security under chosen ciphertext attack.
- Secure options as of 2024:
  - Not post-quantum:
    - RSA-3072 with appropriate padding (e.g., RSA-OAEP) but not basic RSA
    - ElGamal encryption with an appropriate group (finite field or elliptic curve)
  - Post-quantum:
    - ML-KEM (Kyber)

Outline

Security definitions for public key encryption

Security of RSA encryption

Difficulty of factoring

Security of Diffie–Hellman key exchange and Elgamal encryption

Big-O and Little-o Notation

- Let $f(n)$ and $g(n)$ be functions from the positive integers to the positive real numbers.
- (Big-O notation) We write $f(n) = O(g(n))$ if there exists a positive constant c and a positive integer n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$.
  - Example: $3n^3 + 4n^2 + 79 = O(n^3)$.
- (Little-o notation) We write $f(n) = o(g(n))$ if

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

  - Example: $\frac{1}{n} = o(1)$.

Measures of Running Time

Let n be the input and let $\ell = \log_2(n)$ be the input size.

- (Polynomial-time algorithm) One whose worst-case running time function is of the form $O(\ell^c)$, where ℓ is the input size and c is a constant.
- (Exponential-time algorithm) One whose worst-case running time function is of the form $O(2^c\ell)$, where c is a constant.
  - Fully exponential-time algorithms have running time form $2^{c\ell}$, where c is a constant.
  - (Subexponential-time algorithm) One whose worst-case running time function is of the form $2^{o(\ell)}$, and not of the form $O(\ell^c)$ for any constant c.

Roughly speaking, “polynomial-time = efficient”,
“fully exponential-time = terribly inefficient”,
“subexponential-time = inefficient but not terribly so”.

Example (Trial Division)

- Consider the following algorithm (trial division) for factoring RSA-moduli n.
- Trial divide n by the primes $2, 3, 5, \ldots, \sqrt{n}$. If any of these, say l, divides n, then stop and output the factor l of n.
- The running time of this method is at most $\sqrt{n}$ trial divisions, which is $O(\sqrt{n})$.
- Not polynomial time: $\sqrt{n} = 2^{\ell/2}$ where $\ell = \log_2 n$

Subexponential Time

- Let A be an algorithm whose inputs are elements of the integers modulo n, $\mathbb{Z}_n$, or an integer n (so the input size is $\ell = O(\log n)$).
- If the expected running time of A is of the form

$$L_n[\alpha, c] = O\left(\exp\left((c + o(1))(\log n)^{\alpha}(\log\log n)^{1-\alpha}\right)\right),$$

  where c is a positive constant, and α is a constant satisfying $0 < \alpha < 1$, then A is a subexponential-time algorithm.

When $\alpha = 0$: $L_n[0, c] = O((\log n)^{c+o(1)}) = O(\ell^{c+o(1)}$ (polytime).

When $\alpha = 1$: $L_n[1, c] = O(n^{c+o(1)}) = (2^{\ell})^{c+o(1)}$ (fully exponential time).

Special-Purpose Factoring Algorithms

- Examples: Trial division, Pollard's p-1 algorithm, Pollard's ρ algorithm, elliptic curve factoring algorithm, special number field sieve.
- These are only efficient if the number n being factored has a special form (e.g., n has a prime factor p such that $p-1$ has only small factors; or n has a prime factor p that is relatively small).
- To maximize resistance to these factoring attacks on RSA moduli, one should select the RSA primes p and q at random and of the same bitlength.

General-Purpose Factoring Algorithms

- These are factoring algorithms whose running times do not depend of any properties of the number being factored.
- Two major developments in the history of factoring:
  1. (1982) Quadratic sieve factoring algorithm (QS):
     Running time: $L_n[\frac{1}{2}, 1]$.
  2. (1990) Number field sieve factoring algorithm (NFS):
     Running time: $L_n[\frac{1}{3}, 1.923]$.

History of Factoring

Year	Number	Bits	Method	Notes
1903	$2^{67}-1$	67	Naive	F. Cole (3 years of Sundays) In 2010: 0.08 secs in Maple
1988	$\approx 10^{100}$	332	QS	Distributed computation by 100's of computers
1994	RSA-129	425	QS	1600 computers around the world; 8 months
1999	RSA-155	512	NFS	300 workstations + Cray; 5 months
2002	RSA-158	524	NFS	≈ 30 workstations + Cray; 3 months
2003	RSA-174	576	NFS	
2005	RSA-200	663	NFS	(55 years on a single workstation)
2009	RSA-768	768	NFS	2.5 years using several hundred processors
2019	RSA-240	795	NFS	1000 core-years
2020	RSA-250	829	NFS	2700 core-years

Equivalent Security Levels

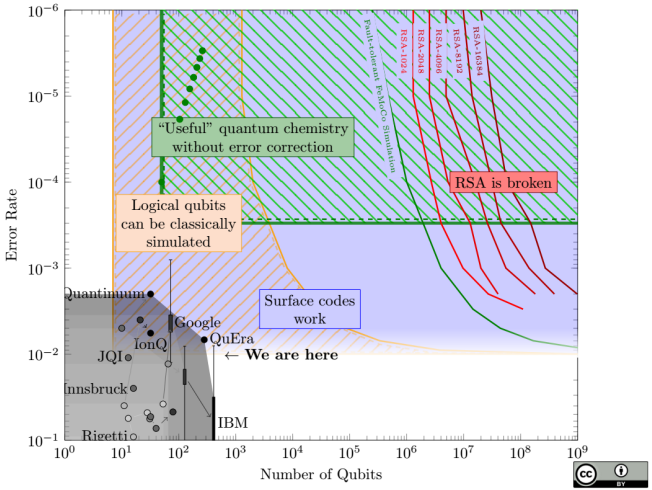
Security in bits	Block cipher	Hash function	RSA $\log_2(n)$
80	SKIPJACK	(SHA-1)	1024
112	Triple-DES	SHA-224	2048
128	AES-128	SHA-256	3072
192	AES-192	SHA-384	7680
256	AES-256	SHA-512	15360

Shor's algorithm

In 1994, Peter Shor published an efficient algorithm for factoring integers using a *quantum computer*.

- Theoretical resources: approximately $O((\log n)^2)$ quantum gates to factor n
- Estimated resources: 2^{24} quantum bits and running for 2^{25} seconds to factor a 2048-bit modulus (`https://arxiv.org/pdf/1902.02332.pdf`)
- Largest quantum computers to date: 1121 qubits (IBM Condor, December 2023); 1180 (Atom Computing, October 2023)

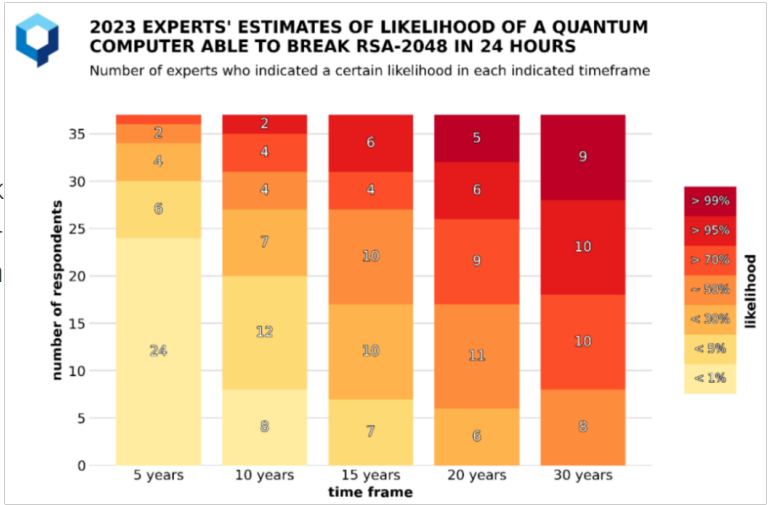
When will we have a cryptographically-relevant quantum computer?



https://sam-jaques.appspot.com/quantum_landscape_2023

When will we have a cryptographically-relevant quantum computer?

$\geq 50\%$ of experts surveyed think there's $\geq 50\%$ chance of a cryptographically relevant quantum computer by 2038



https://globalriskinstitute.org/publication/2023-quantum-threat-timeline-report/

Summary of security of RSA and factoring

- Factoring is believed to be a hard problem for classical computers. However, we have no proof or theoretical evidence that factoring is indeed hard.
- Note however that factoring is known to be easy on a quantum computer. Open question: when will we have a cryptographically-relevant quantum computer?
- 512-bit RSA is considered insecure today.
- 1024-bit RSA should be avoided today.
- Applications have moved to 2048-bit RSA or elliptic curve cryptography
- Applications should start moving to post-quantum algorithms.

Breaking RSA without solving a hard problem

Flawed implementations may be breakable without needing to solve a hard mathematical problem

- Side-channel attacks, which leak information about the secret key during computation
- Bad random number generators, which generate secret keys
- Implementation bugs, which may leak memory contents or do some computations incorrectly

Outline

Security definitions for public key encryption

Security of RSA encryption

Difficulty of factoring

Security of Diffie–Hellman key exchange and Elgamal encryption

DH and Elgamal

Security of Diffie–Hellman key exchange and Elgamal encryption rests on the following two (approximately equivalent) assumptions:

Computational Diffie-Hellman assumption (CDH)

Given g, g^a, g^b, it is computationally infeasible to determine g^{ab}.

Discrete logarithm assumption (DLOG)

Given g and g^a, it is computationally infeasible to determine a.

Are these hard?

Computing discrete logarithms: The random powers method

Suppose we want to find x such that $g^x = 2$.

1. For $i = 1, \ldots, t+1$, choose an integer x_i at random until $g^{x_i} \bmod p$ factors completely using only the first t primes.

$$
\begin{aligned}
g^{x_1} \bmod p &= 2^{e_{1,1}} 3^{e_{1,2}} \cdots p_t^{e_{1,t}} \\
g^{x_2} \bmod p &= 2^{e_{2,1}} 3^{e_{2,2}} \cdots p_t^{e_{2,t}} \\
&\vdots \\
g^{x_{t+1}} \bmod p &= 2^{e_{t+1,1}} 3^{e_{t+1,2}} \cdots p_t^{e_{t+1,t}}
\end{aligned}
$$

2. Take $\log_g$ of both sides:

$$
\begin{aligned}
x_1 &\equiv e_{1,1} \log_g 2 + e_{1,2} \log_g 3 + \cdots + e_{1,t} \log_g p_t \pmod{p-1} \\
x_2 &\equiv e_{2,1} \log_g 2 + e_{2,2} \log_g 3 + \cdots + e_{2,t} \log_g p_t \pmod{p-1} \ldots
\end{aligned}
$$

3. The values x_i and $e_{i,j}$ are known. The values $\log_g p_i$ are unknown. There are $t+1$ equations in t unknowns. Solve!

Computing discrete logarithms

In this way we obtain $\log_g 2, \log_g 3, \log_g 5, \ldots, \log_g p_t$.

- What if we want $\log_g h$ for $h \neq 2, 3, 5, \ldots$?
- Solution: choose an integer r at random until $h \cdot g^r \bmod p$ factors completely using only the first t primes:

$$h \cdot g^r \bmod p = 2^{e_1} 3^{e_2} \cdots p_t^{e_t}$$

- Take $\log_g$ of both sides:

$$\begin{aligned} \log_g(h \cdot g^r) &= \log_g h + r \\ &= e_1 \log_g 2 + e_2 \log_g 3 + \cdots e_t \log_g p_t \pmod{p-1} \end{aligned}$$

- Solve for $\log_g h$.

Status of discrete logarithm methods

- The *index calculus* algorithm can solve discrete logarithms over $\mathbb{Z}_p^*$ in running time

$$L_p\left(\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right) = O\left(\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\log p)^{1/3}(\log\log p)^{2/3}\right)\right)$$

- Equivalent security levels:

Security in bits	Block cipher	Hash function	RSA $\log_2(n)$	DLOG $\log_2(p)$
80	SKIPJACK	(SHA-1)	1024	1024
112	Triple-DES	SHA-224	2048	2048
128	AES-128	SHA-256	3072	3072
192	AES-192	SHA-384	7680	7680
256	AES-256	SHA-512	15360	15360

Summary of discrete logarithm security

- The *index calculus* algorithm can solve discrete logarithms over $\mathbb{Z}_p^*$ in sub-exponential running time

$$L_p\left(\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right)$$

- Discrete logarithm in prime fields and elliptic curve groups is believed to be hard problem for classical computers. However, we have no proof or theoretical evidence that they are indeed hard.
- Shor's algorithm also solves discrete logarithms on a quantum computer.

Key sizes

- As of 2024, we consider something to be infeasible if the best attack takes at least 2^{128} operations.
- For symmetric key primitives usually best attack is brute force key search, so we need keys at least 128 bits long.
- For public key primitives, attack algorithms can take advantage of extra mathematical structure, so we need keys longer than 128 bits in order to achieve 2^{128} difficulty.
- When using multiple algorithms together, need to pick key sizes that match security level (e.g., AES-128 with 3072-bit RSA or 256-bit elliptic curves).