Cryptography and Embedded System Security
CRAESS_I

Xiaolu Hou

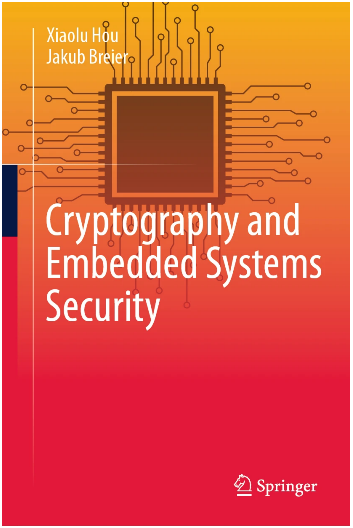
FIIT, STU
xiaolu.hou @ stuba.sk

Course Outline

- Abstract algebra and number theory
- Introduction to cryptography
- Symmetric block ciphers and their implementations
- RSA, RSA signatures, and their implementations
- Probability theory and introduction to SCA
- SPA and non-profiled DPA
- Profiled DPA
- SCA countermeasures
- FA on RSA and countermeasures
- FA on symmetric block ciphers
- FA countermeasures for symmetric block cipher
- Practical aspects of physical attacks
    - Invited speaker: Dr. Jakub Breier, Senior security manager, TTControl GmbH

Recommended reading

- Textbook
  - Sections 3.3, 3.4, 3.5



Lecture Outline

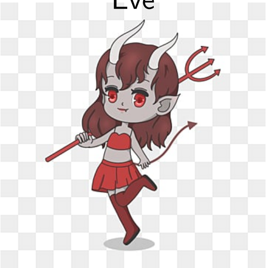
- Introduction
- RSA
- RSA Signatures
- Implementations of Modular Exponentiation
- Implementations of Modular Multiplication

RSA, RSA signatures, and their implementations

- Introduction
- RSA
- RSA Signatures
- Implementations of Modular Exponentiation
- Implementations of Modular Multiplication

Insecure communication channel

Eve

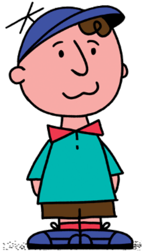


src: https://pngtree.com/

Alice

src: https://www.pngwing.com/en/free-png-zhbsy

Bob



src: https://alicebobstory.com/

Cryptosystem

Definition

A *cryptosystem* is a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ with the following properties.

- $\mathcal{P}$ is a finite set of plaintexts, called *plaintext space*.
- $\mathcal{C}$ is a finite set of ciphertexts, called *ciphertext space*.
- $\mathcal{K}$ is a finite set of keys, called *key space*.
- $\mathcal{E} = \{ E_k : k \in \mathcal{K} \}$, where $E_k : \mathcal{P} \to \mathcal{C}$ is an *encryption function*.
- $\mathcal{D} = \{ D_k : k \in \mathcal{K} \}$, where $D_k : \mathcal{C} \to \mathcal{P}$ is a *decryption function*.
- For each $e \in \mathcal{K}$, there exists $d \in \mathcal{K}$ such that $D_d(E_e(p)) = p$ for all $p \in \mathcal{P}$.

If $e = d$, the cryptosystem is called a *symmetric key cryptosystem*. Otherwise, it is called a *public-key/asymmetric cryptosystem*.

Key exchange

- For symmetric key cipher, a prior communication of the master key (*key exchange*) is required before any ciphertext is transmitted.
- With only a symmetric key cipher, the key exchange may be difficult to achieve due to, e.g. far distance, and too many parties involved.
- In practice, this is where asymmetric key cryptosystem comes into use.
- For example, Alice would like to communicate with Bob using AES.
    - To exchange the master key, k, for AES, she will encrypt k by a public key cryptosystem using Bob's public key e, $c = E_e(k)$.
    - The resulting ciphertext c will be sent to Bob, and Bob can decrypt it with his secret private key d, $k = D_d(c)$.
    - Then Alice and Bob can communicate with key k using AES.

Security of public key cryptosystem

- Clearly, we require that it is computationally infeasible to find the private key d given the public key e.
- In practice, this is guaranteed by some intractable problem
  - A problem is intractable if there does not exist an efficient algorithm to solve it.
- However, the cipher might not be secure in the future.
  - For example, if a quantum computer with enough bits is manufactured, it can break many public key cryptosystems
- A public key cipher is not perfectly secure
  - perfectly secure: in a ciphertext-only attack setting, the attacker cannot obtain any information about the plaintext no matter how much computing power they have.
  - the attacker can brute force the key

Greatest common divisor

Definition

Take $m, n \in \mathbb{Z}$, $m \neq 0$ or $n \neq 0$, the *greatest common divisor* of m and n, denoted $\gcd(m, n)$, is given by $d \in \mathbb{Z}$ such that

- $d > 0$,
- $d|m$, $d|n$, and
- if $c|m$ and $c|n$, then $c|d$.

Example

- All positive divisors of 4 and 6 are $1, 2, 4$ and $1, 2, 3, 6$ respectively. So $\gcd(4, 6) = 2$.
- All the positive divisors of 2 are 1 and 2. All the positive divisors of 3 are 1 and 3. So $\gcd(2, 3) = 1$.

Bézout's identity

Theorem (Bézout's identity)

For any $m, n \in \mathbb{Z}$, such that $m \neq 0$ or $n \neq 0$. $\gcd(m,n)$ exists and is unique. Moreover, $\exists s, t \in \mathbb{Z}$ such that $\gcd(m,n) = sm + tn$.

Example

$$
\begin{aligned}
\gcd(4,6) &= 2 = (-1) \times 4 + 1 \times 6. \\
\gcd(2,3) &= 1 = (-4) \times 2 + 3 \times 3.
\end{aligned}
$$

Euclidean algorithm

Theorem (Euclid's division)

Given $m, n \in \mathbb{Z}$, *take* q, r *such that* $n = qm + r$, *then* $\gcd(m, n) = \gcd(m, r)$.

Thus, to find $\gcd(m, n)$, we can compute Euclid's division repeatedly until we get $r = 0$.

Example

We can calculate $\gcd(120, 35)$ as follows:

$$\begin{aligned} 120 &= 35 \times 3 + 15 && \gcd(120, 35) = \gcd(35, 15), \\ 35 &= 15 \times 2 + 5 && \gcd(35, 15) = \gcd(15, 5), \\ 15 &= 5 \times 3 && \gcd(15, 5) = 5 \implies \gcd(120, 35) = 5. \end{aligned}$$

Euclidean algorithm

Example

We can calculate $\gcd(160, 21)$ using the Euclidean algorithm

$$\begin{aligned}
160 &= 21 \times 7 + 13 & \gcd(160, 21) &= \gcd(21, 13), \\
21 &= 13 \times 1 + 8 & \gcd(21, 13) &= \gcd(13, 8), \\
13 &= 8 \times 1 + 5 & \gcd(13, 8) &= \gcd(8, 5), \\
8 &= 5 \times 1 + 3 & \gcd(8, 5) &= \gcd(5, 3), \\
5 &= 3 \times 1 + 2 & \gcd(5, 3) &= \gcd(3, 2), \\
3 &= 2 \times 1 + 1 & \gcd(3, 2) &= \gcd(2, 1), \\
2 &= 1 \times 2 & \gcd(2, 1) &= 1 \implies \gcd(160, 21) = 1
\end{aligned}$$

Extended Euclidean algorithm

Note

With the intermediate results we have from the Euclidean algorithm, we can also find s, t such that $\gcd(m, n) = sm + tn$ (Bézout's identity).

Example

We have calculated $\gcd(120, 35)$ as follows:

$$
\begin{aligned}
&120 = 35 \times 3 + 15 && \gcd(120, 35) = \gcd(35, 15), \\
&35 = 15 \times 2 + 5 && \gcd(35, 15) = \gcd(15, 5), \\
&15 = 5 \times 3 && \gcd(15, 5) = 5 \implies \gcd(120, 35) = 5.
\end{aligned}
$$

Then

$$
\begin{aligned}
&5 = 35 - 15 \times 2, \\
&15 = 120 - 35 \times 3, \\
&5 = 35 - (120 - 35 \times 3) \times 2 = 120 \times (-2) + 35 \times 7.
\end{aligned}
$$

Extended Euclidean algorithm

Example

We have calculated $\gcd(160, 21)$ using the Euclidean algorithm

$$\begin{array}{ll} 160 = 21 \times 7 + 13 & \gcd(160, 21) = \gcd(21, 13), \\ 21 = 13 \times 1 + 8 & \gcd(21, 13) = \gcd(13, 8), \\ 13 = 8 \times 1 + 5 & \gcd(13, 8) = \gcd(8, 5), \\ 8 = 5 \times 1 + 3 & \gcd(8, 5) = \gcd(5, 3), \\ 5 = 3 \times 1 + 2 & \gcd(5, 3) = \gcd(3, 2), \\ 3 = 2 \times 1 + 1 & \gcd(3, 2) = \gcd(2, 1), \\ 2 = 1 \times 2 & \gcd(2, 1) = 1 \implies \gcd(160, 21) = 1 \end{array}$$

Using the extended Euclidean algorithm, find integers s, t such that $\gcd(160, 21) = s160 + t35$

Extended Euclidean algorithm

Example

By the extended Euclidean algorithm,

$$\begin{aligned} 1 &= 3-2, & 2 &= 5-3, \\ 3 &= 8-5, & 5 &= 13-8, \\ 8 &= 21-13, & 13 &= 160-21\times 7. \end{aligned}$$

We have

$$\begin{aligned} 1 &= 3-(5-3) = 3\times 2 - 5 = 8\times 2 - 5\times 3 = 8\times 2 - (13-8)\times 3 \\ &= 8\times 5 - 13\times 3 = 21\times 5 - 13\times 8 = 21\times 5 - (160-21\times 7)\times 8 \\ &= (-8)\times 160 + 61\times 21. \end{aligned}$$

Prime numbers

Definition

- For $m, n \in \mathbb{Z}$ such that $m \neq 0$ or $n \neq 0$, m and n are said to be *relatively prime/coprime* if $\gcd(m, n) = 1$.
- Given $p \in \mathbb{Z}$, $p > 1$. p is said to be *prime* (or a *prime number*) if for any $m \in \mathbb{Z}$, either m is a multiple of p (i.e. $p|m$) or m and p are coprime (i.e. $\gcd(p, m) = 1$).

Example

- 4 and 9 are relatively prime
- 8 and 6 are not relatively prime
- $2, 3, 5, 7$ are prime numbers
- $6, 9, 21$ are not prime numbers

The Fundamental Theorem of Arithmetic

Theorem (The Fundamental Theorem of Arithmetic)

For any $n \in \mathbb{Z}$, $n > 1$, n can be written in the form

$$n = \prod_{i=1}^{k} p_i^{e_i},$$

where the exponents e_i are positive, the prime numbers $p_1, p_2, \ldots, p_k$ are pairwise distinct and unique up to permutation.

Example

$20 = 2^2 \times 5$, $135 = 3^3 \times 5$.

Congruence class

Definition

For any $a \in \mathbb{Z}$, the *congruence class of* a *modulo* n, denoted $\overline{a}$, is given by

$$\overline{a} := \{ b \mid b \in \mathbb{Z}, b \equiv a \bmod n \} .$$

Lemma

Let $\mathbb{Z}_n$ *denote the set of all congruence classes of* $a \in \mathbb{Z}$ *modulo* n. *Then* $\mathbb{Z}_n = \{ \overline{0}, \overline{1}, \ldots, \overline{n-1} \}$.

Example

Let $n = 5$. We have $\overline{1} = \overline{6} = \overline{-4}$. $\mathbb{Z}_5 = \{ \overline{0}, \overline{1}, \overline{2}, \overline{3}, \overline{4} \}$.

Addition and multiplication in $\mathbb{Z}_n$

Define addition on the set $\mathbb{Z}_n$ as follows:

$$\overline{a} + \overline{b} = \overline{a + b}.$$

Example

- Let $n = 7$, $\overline{3} + \overline{2} = \overline{5}$.
- Let $n = 4$, $\overline{2} + \overline{2} = \overline{4} = \overline{0}$.

Define multiplication on $\mathbb{Z}_n$ as follows

$$\overline{a} \cdot \overline{b} = \overline{ab}.$$

Example

Let $n = 5$,

$$\overline{-2} \cdot \overline{13} = \overline{3} \cdot \overline{3} = \overline{9} = \overline{4}$$

$\mathbb{Z}_n$

Theorem

$(\mathbb{Z}_n, +, \cdot)$, the set $\mathbb{Z}_n$ together with addition multiplication defined just now is a commutative ring.

Remark

For simplicity, we write a instead of $\overline{a}$ and to make sure there is no confusion we would first say $a \in \mathbb{Z}_n$. In particular, $\mathbb{Z}_n = \{ 0, 1, 2, \ldots, n-1 \}$. Furthermore, to emphasize that multiplication or addition is done in $\mathbb{Z}_n$, we write $ab \bmod n$ or $a + b \bmod n$.

Example

Let $n = 5$, we write

$$4 \times 2 \bmod 5 = 8 \bmod 5 = 3, \text{ or } 4 \times 2 \equiv 8 \equiv 3 \bmod 5.$$

$\mathbb{Z}_n$

Lemma

For any $a \in \mathbb{Z}_n$, $a \neq 0$, a has a multiplicative inverse, denoted $a^{-1} \ mod \ n$, if and only if $\gcd(a, n) = 1$.

Corollary

$\mathbb{Z}_n$ is a field if and only if n is prime.

Find multiplicative inverse in $\mathbb{Z}_n$

- Recall that by the extended Euclidean algorithm, we can find integers s, t such that
$$\gcd(a, n) = sa + tn$$
for any $a, n \in \mathbb{Z}$.
- In particular, when $\gcd(a, n) = 1$, we can find s, t such that $1 = as + tn$, which gives $as \bmod n = 1$.
- Thus, we can find $a^{-1} \bmod n = s \bmod n$ by the extended Euclidean algorithm.

Example – Find multiplicative inverse in $\mathbb{Z}_n$

Example

We have calculated $\gcd(160, 21) = 1$ using the Euclidean algorithm. By the extended Euclidean algorithm,

$$
\begin{array}{ll}
1 = 3 - 2, & 2 = 5 - 3, \\
3 = 8 - 5, & 5 = 13 - 8, \\
8 = 21 - 13, & 13 = 160 - 21 \times 7.
\end{array}
$$

We have

$$
\begin{aligned}
1 &= 3 - (5-3) = 3 \times 2 - 5 = 8 \times 2 - 5 \times 3 = 8 \times 2 - (13 - 8) \times 3 \\
&= 8 \times 5 - 13 \times 3 = 21 \times 5 - 13 \times 8 = 21 \times 5 - (160 - 21 \times 7) \times 8 \\
&= (-8) \times 160 + 61 \times 21.
\end{aligned}
$$

Thus

$$21^{-1} \bmod 160 = 61.$$

$\mathbb{Z}_n^*$

Definition

Let $\mathbb{Z}_n^*$ denote the set of congruence classes in $\mathbb{Z}_n$ which have multiplicative inverses:

$$\mathbb{Z}_n^* := \{ a \mid a \in \mathbb{Z}_n, \ \gcd(a, n) = 1 \}.$$

Let $\varphi(n)$ denote the cardinality of $\mathbb{Z}_n^*$

$$\varphi(n) = |\mathbb{Z}_n^*|.$$

φ is called the *Euler's totient function*.

Example

- Let $n = 3$, $\mathbb{Z}_3^* = \{ 1, 2 \}$, $\varphi(3) = 2$.
- Let $n = 4$, $\mathbb{Z}_4^* = \{ 1, 3 \}$, $\varphi(4) = 2$.
- Let $n = p$ be a prime number, $\mathbb{Z}_p^* = \mathbb{Z}_p - \{ 0 \} = \{ 1, 2, \ldots, p - 1 \}$, $\varphi(p) = p - 1$.

Euler's totient function

Theorem

For any $n \in \mathbb{Z}$, $n > 1$,

$$\text{if} \quad n = \prod_{i=1}^{k} p_i^{e_i}, \quad \text{then} \quad \varphi(n) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right), \tag{1}$$

where p_i *are distinct primes.*

Example

- Let $n = 10$. $10 = 2 \times 5$. We can count the elements in $\mathbb{Z}_{10}$ that are coprime to 10 (there are 4 of them): $\mathbb{Z}_{10} = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$. By the above theorem we also have

$$\varphi(10) = 10 \times \left(1 - \frac{1}{2}\right) \times \left(1 - \frac{1}{5}\right) = 4.$$

Euler's totient function

Theorem

For any $n \in \mathbb{Z}$, $n > 1$,

$$\textit{if} \quad n = \prod_{i=1}^{k} p_i^{e_i}, \quad \textit{then} \quad \varphi(n) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right), \tag{2}$$

where p_i *are distinct primes.*

Example

- Let $n = 120$. $120 = 2^3 \times 3 \times 5$.

$$\varphi(120) = ?$$

- Let $n = pq$, where p and q are two distinct primes. Then

$$\varphi(n) = ?$$

Euler's totient function

Example

- Let $n = 120$. $120 = 2^3 \times 3 \times 5$.

$$\varphi(120) = 120 \times \left(1 - \frac{1}{2}\right) \times \left(1 - \frac{1}{3}\right) \times \left(1 - \frac{1}{5}\right) = 32.$$

- Let $n = pq$, where p and q are two distinct primes. Then

$$\varphi(n) = pq\left(1 - \frac{1}{p}\right)\left(1 - \frac{1}{q}\right) = (p-1)(q-1).$$

RSA, RSA signatures, and their implementations

- Introduction
- RSA
- RSA Signatures
- Implementations of Modular Exponentiation
- Implementations of Modular Multiplication

RSA

- Published in 1977
- Named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman.
- RSA is the first public key cryptosystem, and still in use today.
- The security relies on the difficulty of finding the factorization of a composite positive integer.

Definition

Definition (RSA)

Let $n = pq$, where p, q are distinct prime numbers. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, $\mathcal{K} = \mathbb{Z}^*_{\varphi(n)} - \{ 1 \}$. For any $e \in \mathcal{K}$, define encryption

$$E_e : \mathbb{Z}_n \to \mathbb{Z}_n, \quad m \mapsto m^e \bmod n,$$

and the corresponding decryption

$$D_d : \mathbb{Z}_n \to \mathbb{Z}_n, \quad c \mapsto c^d \bmod n,$$

where $d = e^{-1} \bmod \varphi(n)$. The cryptosystem $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{E} = \{ E_e : e \in \mathcal{K} \}$, $\mathcal{D} = \{ D_d : d \in \mathcal{K} \}$, is called *RSA*.

- $\varphi(n) = (p-1)(q-1)$
- Public key: n, e, RSA modulus, encryption exponent
- Private key: d, decryption exponent

Key generation

- Generate randomly and independently two large prime numbers p and q.
- Compute $n = pq$.
  - Normally p and q are supposed to have equal lengths.
  - For example, take p and q to be 512-bit primes, and n will be a 1024-bit modulus.
- Choose $e \in \mathbb{Z}^*_{\varphi(n)}$
  - Note that e is odd since $\varphi(n)$ is even
  - In practice, e is chosen to be small to make the encryption efficient.
  - However, e cannot be too small. It has been shown that only the $n/4$ least significant bits of d suffice to recover d in the case of a small e
- Compute $d = e^{-1} \mod \varphi(n)$ (extended Euclidean algorithm)
  - d cannot be too small, it was proven that if $d < n^{0.292}$, then RSA can be broken

RSA – Example

Example

- As a toy example, suppose Bob would like to generate his private and public keys for RSA.
- Bob randomly generates $p = 3$ and $q = 5$.
- Then he computes $n = 15$ and

$$\varphi(n) = ?$$

$$\mathbb{Z}^*_{\varphi(n)} = ?$$

RSA – Example

Example

- As a toy example, suppose Bob would like to generate his private and public keys for RSA.
- Bob randomly generates $p = 3$ and $q = 5$.
- Then he computes $n = 15$ and

$$\varphi(n) = 2 \times 4 = 8.$$

$$\mathbb{Z}^*_{\varphi(n)} = \{ 1, 3, 5, 7 \}$$

- From $\mathbb{Z}^*_8$, Bob chooses $e = 3$.
- Then by the extended Euclidean algorithm, he computes

$$d = 3^{-1} \bmod 8 = ?$$

RSA – Example

Example

- $p = 3$, $q = 5$, $n = 15$ and $\varphi(n) = 2 \times 4 = 8$.
- From $\mathbb{Z}_8^* = \{1, 3, 5, 7\}$, Bob chooses $e = 3$.
- Then by the extended Euclidean algorithm, he computes

$$8 = 3 \times 2 + 2,\ 3 = 2 \times 1 + 1 \Longrightarrow 1 = 3 - 2 \times 1 = 3 - (8 - 3 \times 2) = -8 + 3 \times 3.$$

- Hence his private key $d = 3^{-1} \bmod 8 = 3$.
- Suppose Alice would like to send plaintext $m = 2$ to Bob, using Bob's public key $n = 15, e = 3$.
- Alice computes

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

- After receiving the ciphertext c from Alice, Bob computes the plaintext using his private key

$$m = c^d \bmod n = ?$$

RSA – Example

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 2 \times 4 = 8, \quad e = 3, \quad d = 3^{-1} \bmod 8 = 3.$$

- Suppose Alice would like to send plaintext $m = 2$ to Bob, using Bob's public key $n = 15, e = 3$.
- Alice computes
$$c = m^e \bmod n = 2^3 \bmod 15 = 8.$$
- After receiving the ciphertext c from Alice, Bob computes the plaintext using his private key
$$m = c^d \bmod n = 8^3 \bmod 15 = 512 \bmod 15 = 2.$$

RSA – Example

Example

$$p = 29, \quad q = 41, \quad n = 1189$$

$$\varphi(n) = ?$$

RSA – Example

Example

$$p = 29, \quad q = 41, \quad n = 1189.$$

$$\varphi(n) = 28 \times 40 = 1120.$$

It is easy to verify that $3 \nmid \varphi(n)$. And we choose $e = 3$. By the extended Euclidean algorithm

$$d = e^{-1} \bmod \varphi(n) = ?.$$

RSA – Example

Example

$$p = 29, \quad q = 41, \quad n = 1189, \quad \varphi(n) = 28 \times 40 = 1120, \quad e = 3.$$

By the extended Euclidean algorithm

$$1120 = 3 \times 373 + 1 \Longrightarrow 1 = 1120 - 3 \times 373.$$

$$d = -373 \bmod 1120 = 747.$$

To send plaintext $m = 2$ to Bob. Alice computes

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

RSA – Example

Example

$$p = 29, \quad q = 41, \quad n = 1189, \quad \varphi(n) = 1120, \quad e = 3, \quad d = -373 \bmod 1120 = 747.$$

To send plaintext $m = 2$ to Bob. Alice computes

$$c = m^e \bmod n = 2^3 \bmod 1189 = 8 \bmod 1189.$$

To decrypt, Bob computes

$$m = c^d \bmod n = 8^{747} \bmod 1189$$

RSA – Example

Example

To decrypt, Bob computes $m = c^d \bmod n = 8^{747} \bmod 1189$.
Since $747 = 512 + 128 + 64 + 32 + 8 + 2 + 1$,

$$8^4 \bmod 1189 = 4096 \bmod 1189 = 529, \quad 8^8 \bmod 1189 = 529^2 \bmod 1189 = 426,$$
$$8^{16} \bmod 1189 = 426^2 \bmod 1189 = 748, \quad 8^{32} \bmod 1189 = 748^2 \bmod 1189 = 674,$$
$$8^{64} \bmod 1189 = 674^2 \bmod 1189 = 78, \quad 8^{128} \bmod 1189 = 78^2 \bmod 1189 = 139,$$
$$8^{256} \bmod 1189 = 139^2 \bmod 1189 = 297, \quad 8^{512} \bmod 1189 = 297^2 \bmod 1189 = 223.$$

$$\begin{aligned}
8^{512+128} \bmod 1189 &= 223 \times 139 \bmod 1189 = 83, \\
8^{64+32} \bmod 1189 &= 78 \times 674 \bmod 1189 = 256 \\
8^{8+2+1} \bmod 1189 &= 426 \times 64 \times 8 \bmod 1189 = 525, \\
8^{747} \bmod 1189 &= 83 \times 256 \times 525 \bmod 1189 = 2.
\end{aligned}$$

A useful lemma

To understand why the decryption works, let us first look at a lemma:

Lemma

Let p be a prime. Then for any $a, b, c \in \mathbb{Z}$ such that $b \equiv c \; mod \; (p-1)$, we have

$$a^b \equiv a^c \; mod \; p.$$

In particular,

$$a^b \equiv a^{b \; mod \; (p-1)} \; mod \; p.$$

Example

Let $p = 5$, $a = 2$, $b = 6$. Then

$$2^6 \equiv ? \bmod 5.$$

A useful lemma

Lemma

Let p be a prime. Then for any $a, b, c \in \mathbb{Z}$ such that $b \equiv c \ mod \ (p-1)$, we have

$$a^b \equiv a^c \ mod \ p.$$

In particular,

$$a^b \equiv a^{b \ mod \ (p-1)} \ mod \ p.$$

Example

Let $p = 5$, $a = 2$, $b = 6$. Then

$$2^6 \equiv 2^{6 \bmod 4} \equiv 2^2 \equiv 4 \bmod 5.$$

We can verify that indeed

$$2^6 \equiv 64 \equiv 4 \bmod 5.$$

Why decryption works

By the choice of e and d,

$$ed \equiv 1 \bmod \varphi(n) \implies ed = \varphi(n)a + 1 \text{ for some } a \in \mathbb{Z}.$$

Then

$$c^d = (m^e)^d = m^{\varphi(n)a+1} = m^{(p-1)(q-1)a} m.$$

By the lemma above:

$$c^d \equiv m \bmod p, \quad c^d \equiv m \bmod q.$$

Chinese Remainder Theorem

Theorem (Chinese Remainder Theorem)

Let $m_1, m_2, \ldots, m_k$ be pairwise coprime integers. For any $a_1, a_2, \ldots, a_k \in \mathbb{Z}$, the system of simultaneous congruences

$$x \equiv a_1 \textit{ mod } m_1, \quad x \equiv a_2 \textit{ mod } m_2, \quad \ldots \quad x \equiv a_k \textit{ mod } m_k$$

has a unique solution modulo $m = \prod_{i=1}^{k} m_i$.

Example

Take two distinct primes p, q, and let $n = pq$. By CRT, for any $a \in \mathbb{Z}_n$, there is a unique solution $x \in \mathbb{Z}_n$ such that

$$x \equiv a \bmod p, \quad x \equiv a \bmod q.$$

Since $a \equiv a \bmod p$ and $a \equiv a \bmod q$, the unique solution is given by $x = a \in \mathbb{Z}_n$.

Why decryption works

By the choice of e and d,

$$ed \equiv 1 \bmod \varphi(n) \Longrightarrow ed = \varphi(n)a + 1 \text{ for some } a \in \mathbb{Z}.$$

Then

$$c^d = (m^e)^d = m^{\varphi(n)a+1} = m^{(p-1)(q-1)a}m.$$

By the lemma above:

$$c^d \equiv m \bmod p, \quad c^d \equiv m \bmod q.$$

By Chinese Remainder Theorem,

$$c^d \equiv m \bmod n.$$

Security of RSA

- If p or q is known to the attacker
  - can factorize n and compute $\varphi(n)$
  - with e, d can be computed using the extended Euclidean algorithm
- All $p, q, \varphi(n)$ should be kept secret
- Of course, if the attacker can factorize n with an efficient algorithm, then RSA is broken.
  - Up to now, the best-known algorithm for integer factorization has been used to factorize RSA modulus of bit length 768
  - In practice, the most commonly used RSA modulus n is 1024, 2048, or 4096 bit.
  - On the other hand, there is no proof that factorizing an integer n is infeasible.
- It is not proven that RSA is secure if factoring is computationally infeasible – there might be other ways to attack RSA.

RSA, RSA signatures, and their implementations

Digital signatures

- Digital signatures provide means for an entity to bind its identity to a message.
- This normally means that the sender uses their private key to sign the (hashed) message.
- Whoever has access to the public key can then verify the origin of the message.
- For example, the message can be electronic contracts or electronic bank transactions.
- Suppose Alice signs a message m with a private key d and generates signature s.
- Bob receives the message and the signature, he can then verify s with public key e and a *verification algorithm*.
- Given m and s, the verification algorithm returns true to indicate a valid signature and false otherwise.

RSA signatures

- To use RSA for digital signature, we again let p and q be two distinct primes.
- $n = pq$, choose $e \in \mathbb{Z}^*_{\varphi(n)}$, compute $d = e^{-1} \bmod \varphi(n)$.
- The public key consists of e and n.
- d is the private key.
- p, q and $\varphi(n)$ should be kept secret.

RSA signatures

To sign a message m, Alice computes the signature

$$s = m^d \bmod n.$$

Then Alice sends both m and s to Bob. To verify the signature, Bob computes

$$s^e \bmod n.$$

If $s \equiv m \bmod n$, then the verification algorithm outputs true, and false otherwise.

- Up to now, the only method known to compute s from $m \bmod n$ is using d, so if the verification algorithm outputs true, Bob can conclude that Alice is the owner of d.

RSA signatures – Example

Example

Alice chooses $p = 5$ and $q = 7$. Then $n = 35$ and

$$\varphi(n) = ?.$$

RSA signatures – Example

Example

- Alice chooses $p = 5$ and $q = 7$.
- Then
$$n = 35, \quad \varphi(n) = 24$$
- Suppose Alice chooses $e = 5$, which is coprime to 24.
- By the extended Euclidean algorithm
$$d = e^{-1} \bmod \varphi(n) = ?$$

RSA signatures – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5.$$

- By the extended Euclidean algorithm

$$24 = 5 \times 4 + 4,\ 5 = 4 + 1 \Longrightarrow 1 = 5 - (24 - 5 \times 4) = 24 \times (-1) + 5 \times 5,$$

  we have $d = e^{-1} \bmod 24 = 5$.
- To sign message $m = 10$, Alice computes

$$s = m^d \bmod n = ?$$

- Alice sends both the message and signature to Bob.
- Bob verifies the signature

$$s^e \bmod n = ?$$

RSA signatures – Example

Example

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5$$

- To sign message $m = 10$, Alice computes

$$s = m^d \bmod n = 10^5 \bmod 35 = 5.$$

- Alice sends both the message $m = 10$ and signature $s = 5$ to Bob.
- Bob verifies the signature

$$s^e \bmod n = 5^5 \bmod 35 = 10 = m.$$

Forgery attack on RSA signatures

- The most common attack for a digital signature is to create a valid signature for a message without knowing the secret key.
- Such an attack is called *forgery*
- Suppose the attacker, Eve, knows messages m_1, m_2 and their corresponding signatures s_1 and s_2.
- Eve computes $s = s_1 s_2 \bmod n$ and $m = m_1 m_2 \bmod n$.
- Since
$$s = m_1^d m_2^d \bmod n = (m_1 m_2)^d \bmod n = m^d \bmod n,$$
s is a valid signature for m.
- RSA signatures are commonly used together with a fast public hash function h

Hash functions

- Hash functions map data of arbitrary length to a binary array of some fixed length called *hash values* or *message digests*
- The following are the properties that should be met in a properly designed cryptographic hash function:
  (a) it is quick to compute a hash-value for any given input;
  (b) it is computationally infeasible to generate an input that yields a given hash value (a preimage);
  (c) it is computationally infeasible to find a second input that maps to the same hash value when one input is already known (a second preimage);
  (d) it is computationally infeasible to find any pair of different messages that produce the same hash value (a collision).

RSA signature with hash function

- To sign a message m, Alice computes the signature

$$s = h(m)^d \bmod n.$$

- Then she sends both m and s to Bob.
- Bob computes $s^e \bmod n$ and $h(m)$.
- If $s^e \bmod n = h(m)$, then Bob concludes the signature is valid.

Forgery attack

- Suppose the attacker, Eve, knows messages m_1, m_2 and their corresponding signatures s_1 and s_2.

Without hash function

- Eve computes $s = s_1 s_2 \mod n$ and $m = m_1 m_2 \mod n$.
- Since
$$s = m_1^d m_2^d \mod n = (m_1 m_2)^d \mod n = m^d \mod n,$$
s is a valid signature for m.

With hash function

- She can compute $h(m_1)$ and $h(m_2)$ as h is public.
- However, to repeat the forgery attack, she needs to find m such that $h(m) = h(m_1)h(m_2)$, which is computationally infeasible according to property (b) of hash functions

- Introduction
- RSA
- RSA Signatures
- Implementations of Modular Exponentiation
- Implementations of Modular Multiplication

Modular exponentiation

- To implement RSA or RSA signatures, we need to compute $a^d \bmod n$ for some integer $a \in \mathbb{Z}_n$,
- $n = pq$ is a product of two distinct primes and $d \in \mathbb{Z}^*_{\varphi(n)}$.
- We can compute $d - 1$ modular multiplications.
  - inefficient for large d
  - impossible for practical values of d – bit length more than 1000
- Two methods
  - square and multiply algorithm
  - CRT-based RSA implementation

Square and multiply algorithm

- Let $n \geq 2$ be an integer, $d \in \mathbb{Z}_{\varphi(n)}$, $a \in \mathbb{Z}_n$
- Binary representation of $d = d_{\ell_d-1} \dots d_2 d_1 d_0$, where $d_i = 0, 1$ and

$$d = \sum_{i=0}^{\ell_d-1} d_i 2^i.$$

- We have

$$a^d = a^{\sum_{i=0}^{\ell_d-1} d_i 2^i} = \prod_{i=0}^{\ell_d-1} (a^{2^i})^{d_i} = \prod_{0 \leq i < \ell_d, d_i=1} a^{2^i}.$$

- Thus, to compute $a^d \bmod n$, we can
  - First compute a^{2^i} for $0 \leq i < \ell_d$
  - Then a^d is a product of a^{2^i} for which $d_i = 1$
- Compared to $d - 1$ modular multiplications, this requires at most $2 \log_2 d$ multiplications

Square and multiply algorithm

Algorithm 1: Right-to-left square and multiply algorithm for computing modular exponentiation

Input: n, a, d// $n \in \mathbb{Z}, n \geq 2$; $a \in \mathbb{Z}_n$; $d \in \mathbb{Z}_{\varphi(n)}$ has bit length ℓ_d

Output: $a^d \bmod n$

result $= 1$, $t = a$

for $i = 0$, $i < \ell_d$, $i++$ **do**

    // ith bit of d is 1

    if $d_i = 1$ **then**

        // mutiply by a^{2^i}

        result = result $* \, t \bmod n$// $a^d = \prod_{0 \leq i < \ell_d, d_i = 1} a^{2^i}$

    // $t = a^{2^{i+1}}$

    $t = t * t \bmod n$

return result

Right-to-left square and multiply algorithm

result $= 1$, $t = a$
for $i = 0$, $i < \ell_d$, $i++$ **do**
  // ith bit of d is 1
  if $d_i = 1$ **then**
    // mutiply by a^{2^i}
    result $=$ result$*t \bmod n$
  // $t = a^{2^{i+1}}$
  $t = t * t \bmod n$
return result

Example

Let $n = 15,\ d = 3 = 11_2,\ a = 2.$ Then

$$a^d \bmod n = 2^3 \bmod 15 = 8 \bmod 15 = 8$$

i	d_i	t	result
0	?	?	?
1	?	?	?

Right-to-left square and multiply algorithm

```
result = 1, t = a
for i = 0, i < ℓ_d, i++ do
    // ith bit of d is 1
    if d_i = 1 then
        // mutiply by a^(2^i)
        result = result*t mod n
    // t = a^(2^(i+1))
    t = t * t mod n
return result
```

Example

Let $n = 15,\ d = 3 = 11_2,\ a = 2.$ Then

$$a^d \bmod n = 2^3 \bmod 15 = 8 \bmod 15 = 8$$

i	d_i	t	result
0	1	4	2
1	1	1	8

Right-to-left square and multiply algorithm

result $= 1$, $t = a$
for $i = 0,\ i < \ell_d,\ i++$ **do**
    // ith bit of d is 1
    if $d_i = 1$ **then**
        // mutiply by a^{2^i}
        result $=$ result$*t \bmod n$
    // $t = a^{2^{i+1}}$
    $t = t * t \bmod n$
return result

Example

Let $n = 23,\ d = 4 = 100_2,\ a = 5.$ Then

$$a^d \bmod n = 5^4 \bmod 23 = 625 \bmod 23 = 4$$

i	d_i	t	result
0	?	?	?
1	?	?	?
2	?	?	?

Right-to-left square and multiply algorithm

```
result = 1, t = a
for i = 0, i < ℓ_d, i++ do
    // ith bit of d is 1
    if d_i = 1 then
        // mutiply by a^(2^i)
        result = result*t mod n
    // t = a^(2^(i+1))
    t = t*t mod n
return result
```

Example

Let $n = 23,\ d = 4 = 100_2,\ a = 5.$ Then

$$a^d \bmod n = 5^4 \bmod 23 = 625 \bmod 23 = 4$$

i	d_i	t	result
0	0	2	1
1	0	4	1
2	1	16	4

Left-to-right square and multiply algorithm

Algorithm 2: Left-to-right square and multiply algorithm for computing modular exponentiation.

Input: n, a, d // $n \in \mathbb{Z}, n \geq 2$; $a \in \mathbb{Z}_n$; $d \in \mathbb{Z}_{\varphi(n)}$

Output: $a^d \bmod n$

$t = 1$

for $i = \ell_d - 1$, $i \geq 0$, $i - -$ **do**

    $t = t * t \bmod n$

    // ith bit of d is 1

    if $d_i = 1$ **then**

        $t = a * t \bmod n$

return t

Left-to-right square and multiply algorithm

$t = 1$
for $i = \ell_d - 1,\ i \geq 0,\ i--$ **do**
  $t = t * t \bmod n$
  // ith bit of d is 1
  if $d_i = 1$ **then**
    $t = a * t \bmod n$
return t

Example

Let $n = 15,\ d = 3 = 11_2,\ a = 2.$ Then

$$a^d \bmod n = 2^3 \bmod 15 = 8 \bmod 15 = 8$$

i	d_i	t
1	?	?
0	?	?

Left-to-right square and multiply algorithm

$t = 1$
for $i = \ell_d - 1,\ i \geq 0,\ i--$ **do**
    $t = t * t \bmod n$
    // ith bit of d is 1
    if $d_i = 1$ **then**
        $t = a * t \bmod n$
return t

Example

Let $n = 15,\ d = 3 = 11_2,\ a = 2.$ Then

$$a^d \bmod n = 2^3 \bmod 15 = 8 \bmod 15 = 8$$

i	d_i	t
1	1	2
0	1	8

Left-to-right square and multiply algorithm

$t = 1$
for $i = \ell_d - 1,\ i \geq 0,\ i - -$ **do**
    $t = t * t \bmod n$
    // ith bit of d is 1
    if $d_i = 1$ **then**
        $t = a * t \bmod n$
return t

Example

Let $n = 23,\ d = 4 = 100_2,\ a = 5.$ Then

$$a^d \bmod n = 5^4 \bmod 23 = 625 \bmod 23 = 4$$

i	d_i	t
2	?	?
1	?	?
0	?	?

Left-to-right square and multiply algorithm

$t = 1$
for $i = \ell_d - 1,\ i \geq 0,\ i--$ **do**
  $t = t * t \bmod n$
  // ith bit of d is 1
  if $d_i = 1$ **then**
    $t = a * t \bmod n$
return t

Example

Let $n = 23,\ d = 4 = 100_2,\ a = 5.$ Then

$$a^d \bmod n = 5^4 \bmod 23 = 625 \bmod 23 = 4$$

i	d_i	t
2	1	5
1	0	2
0	0	4

CRT-based

- p, q: distinct primes
- $n = pq$ is the RSA modulus
- $d \in \mathbb{Z}^*_{\varphi(n)}$ is the private key for RSA or RSA signatures.

Chinese Remainder Theorem

Theorem (Chinese Remainder Theorem)

Let $m_1, m_2, \ldots, m_k$ be pairwise coprime integers. For any $a_1, a_2, \ldots, a_k \in \mathbb{Z}$, the system of simultaneous congruences

$$x \equiv a_1 \textit{ mod } m_1, \quad x \equiv a_2 \textit{ mod } m_2, \quad \ldots \quad x \equiv a_k \textit{ mod } m_k$$

has a unique solution modulo $m = \prod_{i=1}^{k} m_i$.

Example

Take two distinct primes p, q, and let $n = pq$. By CRT, for any $a \in \mathbb{Z}$, there is a unique solution $x \in \mathbb{Z}_n$ such that

$$x \equiv a \bmod p, \quad x \equiv a \bmod q.$$

$\Longrightarrow$ to find solution for $x \equiv a^d \bmod n$ is equivalent to solving

$$x \equiv a^d \bmod p, \quad x \equiv a^d \bmod q.$$

A useful lemma

Lemma

Let p be a prime. Then for any $a, b, c \in \mathbb{Z}$ such that $b \equiv c \ mod\ (p-1)$, we have

$$a^b \equiv a^c \ mod\ p.$$

In particular,

$$a^b \equiv a^{b \ mod\ (p-1)} \ mod\ p.$$

Example

Let $p = 5$, $a = 2$, $b = 6$. Then

$$2^6 \equiv 2^{6 \bmod 4} \equiv 2^2 \equiv 4 \bmod 5.$$

We can verify that indeed

$$2^6 \equiv 64 \equiv 4 \bmod 5.$$

CRT-based

By the Chinese Remainder Theorem, finding the solution for $x \equiv a^d \bmod n$ is equivalent to solving

$$x \equiv a^d \bmod p, \quad x \equiv a^d \bmod q.$$

By the lemma, we can compute

$$x_p := a^{d \bmod (p-1)} \bmod p, \quad x_q := a^{d \bmod (q-1)} \bmod q,$$

and solve for

$$x \equiv x_p \bmod p, \quad x \equiv x_q \bmod q.$$

An implementation that computes $a^d \bmod n$ by solving the above equation is called *CRT-based RSA*.

Gauss's algorithm

We have discussed in week 1 that, we can compute

$$M_q = q,\ M_p = p,\ y_q = M_q^{-1} \bmod p = q^{-1} \bmod p,\ y_p = M_p^{-1} \bmod q = p^{-1} \bmod q,$$

and

$$x = x_p y_q q + x_q y_p p \bmod n$$

gives us the solution to

$$x \equiv x_p \bmod p, \quad x \equiv x_q \bmod q.$$

Calculating x by with this method is called the *Gauss's algorithm*.

Garner's algorithm

Garner's algorithm calculates

$$x = x_p + ((x_q - x_p)y_p \bmod q)p.$$

This indeed gives the solution to

$$x \equiv x_p \bmod p, \quad x \equiv x_q \bmod q.$$

Firstly, it is straightforward to see $x \equiv x_p \bmod p$. Furthermore,

$$x \equiv x_p + (x_q - x_p) \equiv x_q \bmod q.$$

Since $x_p \in \mathbb{Z}_p$, $x_p < p$. Similarly, $(x_q - x_p)y_p \bmod q \leq q - 1$. And

$$x = x_p + ((x_q - x_p)y_p \bmod q)p < p + (q - 1)p = n,$$

thus $x \in \mathbb{Z}_n$.

CRT-based RSA implementation – Example

CRT-based RSA implementation

$$x_p := a^{d \bmod (p-1)} \bmod p, \quad x_q := a^{d \bmod (q-1)} \bmod q,$$

$$M_q = q, \quad M_p = p$$

$$y_q = M_q^{-1} \bmod p = q^{-1} \bmod p, \quad y_p = M_p^{-1} \bmod q = p^{-1} \bmod q.$$

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 8, \quad e = 3, \quad d = 3, \quad m = 2, \quad c = 8$$

After receiving the ciphertext c, Bob computes the plaintext using his private key

$$m = c^d \bmod n = 8^3 \bmod 15 = 512 \bmod 15 = 2 \bmod 15.$$

With CRT-based RSA implementation, Bob computes

$$m_p =? \quad m_q =? \quad y_p =? \quad y_q =?$$

CRT-based RSA implementation – Example

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 8, \quad e = 3, \quad d = 3, \quad m = 2, \quad c = 8$$

After receiving the ciphertext c, with CRT-based RSA implementation, Bob computes

$$\begin{aligned} m_p &= c^{d \bmod (p-1)} \bmod p = 8^{3 \text{ mod } 2} \bmod 3 = 8 \bmod 3 = 2, \\ m_q &= c^{d \bmod (q-1)} \bmod q = 8^{3 \text{ mod } 4} \bmod 5 = 512 \bmod 5 = 2. \end{aligned}$$

By the extended Euclidean algorithm,

$$5 = 3 \times 1 + 2, \ 3 = 2 + 1 \Longrightarrow 1 = 3 - (5 - 3) = 3 \times 2 - 5.$$

Thus

$$\begin{aligned} y_p &= p^{-1} \bmod q = 3^{-1} \bmod 5 = 2 \bmod 5, \\ y_q &= q^{-1} \bmod p = 5^{-1} \bmod 3 = -1 \bmod 3 = 2 \bmod 3. \end{aligned}$$

CRT-based RSA implementation – Example

Gauss's algorithm

$$x = x_p y_q q + x_q y_p p \bmod n$$

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 8, \quad e = 3, \quad d = 3, \quad c = 8$$

With CRT-based RSA implementation, Bob computes

$$m_p = 2 \quad m_q = 2 \quad y_p = 2 \quad y_q = 2.$$

By Gauss's algorithm

$$m = ?$$

CRT-based RSA implementation – Example

Gauss's algorithm

$$x = x_p y_q q + x_q y_p p \bmod n$$

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 8, \quad e = 3, \quad d = 3, \quad m = 2, \quad c = 8$$

With CRT-based RSA implementation, Bob computes

$$m_p = 2 \quad m_q = 2 \quad y_p = 2 \quad y_q = 2.$$

By Gauss's algorithm

$$m = m_p y_q q + m_q y_p p \bmod n = 2 \times 2 \times 5 + 2 \times 2 \times 3 \bmod 15 = 32 \bmod 15 = 2.$$

CRT-based RSA implementation – Example

Garner's algorithm

$$x = x_p + ((x_q - x_p)y_p \bmod q)p.$$

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 8, \quad e = 3, \quad d = 3, \quad c = 8$$

With CRT-based RSA implementation, Bob computes

$$m_p = 2 \quad m_q = 2 \quad y_p = 2 \quad y_q = 2.$$

By Garner's algorithm

$$m =?$$

CRT-based RSA implementation – Example

Garner's algorithm

$$x = x_p + ((x_q - x_p)y_p \bmod q)p.$$

Example

$$p = 3, \quad q = 5, \quad n = 15, \quad \varphi(n) = 8, \quad e = 3, \quad d = 3, \quad c = 8$$

With CRT-based RSA implementation, Bob computes

$$m_p = 2 \quad m_q = 2 \quad y_p = 2 \quad y_q = 2.$$

By Garner's algorithm

$$m = m_p + ((m_q - m_p)y_p \bmod q)p = 2 + 0 = 2.$$

CRT-based RSA implementation – Example

CRT-based RSA implementation

$$x_p := a^{d \bmod (p-1)} \bmod p, \qquad x_q := a^{d \bmod (q-1)} \bmod q,$$

$$M_q = q, \quad M_p = p,$$

$$y_q = M_q^{-1} \bmod p = q^{-1} \bmod p, \quad y_p = M_p^{-1} \bmod q = p^{-1} \bmod q,$$

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

To sign message $m = 10$, with CRT-based RSA implementation, Alice computes

$$s_p = ? \quad s_q = ? \quad y_p = ? \quad y_q = ?$$

CRT-based RSA implementation – Example

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

To sign message $m = 10$, with CRT-based RSA implementation, Alice computes

$$\begin{aligned} s_p &= m^{d \bmod (p-1)} \bmod p = 10^{5 \text{ mod } 4} \bmod 5 = 0, \\ s_q &= m^{d \bmod (q-1)} \bmod q = 10^{5 \text{ mod } 6} \bmod 7 = 5. \end{aligned}$$

By the extended Euclidean algorithm

$$7 = 5 + 2,\ 5 = 2 \times 2 + 1 \Longrightarrow 1 = 5 - 2 \times (7 - 5) = 5 \times 3 - 2 \times 7$$

We have

$$\begin{aligned} y_p &= p^{-1} \bmod q = 3 \bmod 7, \\ y_q &= q^{-1} \bmod p = -2 \bmod 5 = 3. \end{aligned}$$

CRT-based RSA implementation – Example

Gauss's algorithm

$$x = x_p y_q q + x_q y_p p \bmod n$$

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

With CRT-based RSA implementation, Alice computes

$$s_p = 0 \quad s_q = 5 \quad y_p = 3 \quad y_q = 3.$$

By Gauss's algorithm

$$s = ?$$

CRT-based RSA implementation – Example

Gauss's algorithm

$$x = x_p y_q q + x_q y_p p \bmod n$$

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

With CRT-based RSA implementation, Alice computes

$$s_p = 0 \quad s_q = 5 \quad y_p = 3 \quad y_q = 3.$$

By Gauss's algorithm

$$s = s_p y_q q + s_q y_p p \bmod n = 5 \times 3 \times 5 \bmod 35 = 5.$$

CRT-based RSA implementation – Example

Garner's algorithm

$$x = x_p + ((x_q - x_p)y_p \bmod q)p.$$

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

With CRT-based RSA implementation, Alice computes

$$s_p = 0 \quad s_q = 5 \quad y_p = 3 \quad y_q = 3.$$

By Garner's algorithm

$$s = ?$$

CRT-based RSA implementation – Example

Garner's algorithm

$$x = x_p + ((x_q - x_p)y_p \bmod q)p.$$

Example (RSA signature computation)

$$p = 5, \quad q = 7, \quad n = 35, \quad \varphi(n) = 24, \quad e = 5, \quad d = 5.$$

With CRT-based RSA implementation, Alice computes

$$s_p = 0 \quad s_q = 5 \quad y_p = 3 \quad y_q = 3.$$

By Garner's algorithm

$$s = s_p + ((s_q - s_p)y_p \bmod q)p = 0 + (5 \times 3 \bmod 7) \times 5 = 1 \times 5 = 5.$$

CRT-based RSA implementation – Example

CRT-based RSA implementation

$$x_p := a^{d \bmod (p-1)} \bmod p, \qquad x_q := a^{d \bmod (q-1)} \bmod q,$$

$$M_q = q, \quad M_p = p,$$

$$y_q = M_q^{-1} \bmod p = q^{-1} \bmod p, \quad y_p = M_p^{-1} \bmod q = p^{-1} \bmod q,$$

Example

$$p = 29, \quad q = 41, \quad n = 1189, \quad \varphi(n) = 1120, \quad e = 3, \quad d = 747, \quad c = 8$$

With CRT-based RSA implementation, Bob computes

$$m_p =? \quad m_q =? \quad y_p =? \quad y_q =?$$

CRT-based RSA implementation – Example

Example

$$p = 29, \quad q = 41, \quad n = 1189, \quad \varphi(n) = 1120, \quad e = 3, \quad d = 747, \quad c = 8$$

With CRT-based RSA implementation, Bob computes

$$\begin{aligned} m_p &= c^{d \bmod (p-1)} \bmod p = 8^{747 \textsf{ mod } 28} \bmod 29 = 8^{19} \bmod 29 = 2, \\ m_q &= c^{d \bmod (q-1)} \bmod q = 8^{747 \textsf{ mod } 40} \bmod 41 = 8^{27} \bmod 41 = 2. \end{aligned}$$

By the extended Euclidean algorithm

$$\begin{aligned} & 41 = 29 + 12,\ 29 = 12 \times 2 + 5,\ 12 = 5 \times 2 + 2,\ 5 = 2 \times 2 + 1, \\ 1 &= 5 - 2 \times (12 - 5 \times 2) = -2 \times 12 + (29 - 12 \times 2) \times 5 \\ &= 29 \times 5 - 12 \times (41 - 29) = -41 \times 12 + 29 \times 17. \\ y_p &= p^{-1} \bmod q = 29^{-1} \bmod 41 = 17 \bmod 41, \\ y_q &= q^{-1} \bmod p = 41^{-1} \bmod 29 = -12 \bmod 29 = 17 \bmod 29. \end{aligned}$$

CRT-based RSA implementation – Example

Gauss's algorithm

$$x = x_p y_q q + x_q y_p p \bmod n$$

Example

$$p = 29, \quad q = 41, \quad n = 1189, \quad \varphi(n) = 1120, \quad e = 3, \quad d = 747, \quad c = 8$$

With CRT-based RSA implementation, Bob computes

$$m_p = 2 \quad m_q = 2 \quad y_p = 17 \quad y_q = 17$$

By Gauss's algorithm

$$m =?$$

CRT-based RSA implementation – Example

Gauss's algorithm

$$x = x_p y_q q + x_q y_p p \bmod n$$

Example

$$p = 29, \quad q = 41, \quad n = 1189, \quad \varphi(n) = 1120, \quad e = 3, \quad d = 747, \quad c = 8$$

With CRT-based RSA implementation, Bob computes

$$m_p = 2 \quad m_q = 2 \quad y_p = 17 \quad y_q = 17$$

By Gauss's algorithm

$$m = m_p y_q q + m_q y_p p \bmod n = 2\times17\times41 + 2\times17\times29 \bmod 1189 = 2380 \bmod 1189 = 2.$$

CRT-based RSA implementation – Example

Garner's algorithm

$$x = x_p + ((x_q - x_p)y_p \bmod q)p.$$

Example

$$p = 29, \quad q = 41, \quad n = 1189, \quad \varphi(n) = 1120, \quad e = 3, \quad d = 747, \quad c = 8$$

With CRT-based RSA implementation, Bob computes

$$m_p = 2 \quad m_q = 2 \quad y_p = 17 \quad y_q = 17$$

By Garner's algorithm

$$m = ?$$

CRT-based RSA implementation – Example

Garner's algorithm

$$x = x_p + ((x_q - x_p)y_p \bmod q)p.$$

Example

$$p = 29, \quad q = 41, \quad n = 1189, \quad \varphi(n) = 1120, \quad e = 3, \quad d = 747, \quad c = 8$$

With CRT-based RSA implementation, Bob computes

$$m_p = 2 \quad m_q = 2 \quad y_p = 17 \quad y_q = 17$$

By Garner's algorithm

$$m = m_p + ((m_q - m_p)y_p \bmod q)p = 2 + 0 = 2.$$

CRT-based RSA implementation

- y_p and y_q can be precomputed, which saves time during communication.
- The intermediate values during the computation are only half as big compared to computations of $a^d \bmod n$ since they are in $\mathbb{Z}_p$ or $\mathbb{Z}_q$ rather than $\mathbb{Z}_n$.
- $x_p = a^{d \bmod (p-1)} \bmod p$ and $x_q = a^{d \bmod (q-1)} \bmod q$ can be calculated by square and multiply algorithm to further improve the efficiency.
- $d \bmod (p-1)$ and $d \bmod (q-1)$ are much smaller than d, computing x_p or x_q requires fewer multiplications than computing $a^d \bmod p$ and $a^d \bmod q$.
- Compared to Gauss's algorithm, Garner's algorithm does not require the final modulo n reduction.

RSA, RSA signatures, and their implementations

- Introduction
- RSA
- RSA Signatures
- Implementations of Modular Exponentiation
- Implementations of Modular Multiplication

Modular operations

- To have more efficient modular exponentiation implementations, we need to compute modular addition, subtraction, inverse, and multiplications.
- For modular addition and subtraction, we can just compute the corresponding integer operation and then perform a single reduction modulo the modulus.
- For inverse modulo an integer, as has been mentioned a few times, we can utilize the extended Euclidean algorithm.
- We will look at one method for computing modular multiplication

Notations

- n: an integer of bit length ℓ_n,

$$2^{\ell_n - 1} \leq n < 2^{\ell_n}.$$

- $a, b \in \mathbb{Z}_n$, in particular, $0 \leq a, b < n$.
- ω: the computer's word size
  - for a 64-bit processor, the *word size* is 64
- Let $\kappa = \lceil \ell_n / \omega \rceil$, i.e. $(\kappa - 1)\omega < \ell_n \leq \kappa\omega$.
- Then ($||$ indicates concatenation, $0 \leq a_i < 2^\omega$)

$$a = a_{\kappa-1} || a_{\kappa-2} || \ldots || a_0,$$

- Note that some a_i might be 0 if the bit length of a is less than ℓ_n. We have

$$a = \sum_{i=0}^{\kappa-1} a_i (2^\omega)^i.$$

Notations

- n: an integer of bit length ℓ_n
- $a, b \in \mathbb{Z}_n$, in particular, $0 \leq a, b < n$.
- ω: the computer's word size
- Let $\kappa = \lceil \ell_n/\omega \rceil$, i.e. $(\kappa - 1)\omega < \ell_n \leq \kappa\omega$.
- Then ($||$ indicates concatenation, $0 \leq a_i < 2^\omega$)

$$a = a_{\kappa-1}||a_{\kappa-2}||\ldots||a_0,$$

- Note that some a_i might be 0 if the bit length of a is less than ℓ_n. We have

$$a = \sum_{i=0}^{\kappa-1} a_i (2^\omega)^i.$$

Example

$\omega = 2$, $a = 13 = 1101_2$, $n = 15$. Then

$$\ell_n = 4, \quad \kappa = \lceil \ell_n/\omega \rceil = \lceil 4/2 \rceil = 2.$$

Notations – Example

- n: an integer of bit length ℓ_n
- $a, b \in \mathbb{Z}_n$, in particular, $0 \leq a, b < n$.
- Let $\kappa = \lceil \ell_n / \omega \rceil$
- Then ($||$ indicates concatenation, $0 \leq a_i < 2^\omega$)

$$a = a_{\kappa-1} || a_{\kappa-2} || \dots || a_0,$$

- Note that some a_i might be 0 if the bit length of a is less than ℓ_n. We have

$$a = \sum_{i=0}^{\kappa-1} a_i (2^\omega)^i.$$

Example

$\omega = 2$, $a = 13 = 1101_2$, $n = 15$. Then $\ell_n = 4$, $\kappa = 2$.

$$a_0 = 01_2 = 1, \quad a_1 = 11_2 = 3, \quad a = a_0 (2^\omega)^0 + a_1 (2^\omega)^1 = 1 + 3 \times 4 = 13$$

Notations – Example

- n: an integer of bit length ℓ_n
- $a, b \in \mathbb{Z}_n$, in particular, $0 \leq a, b < n$.
- Let $\kappa = \lceil \ell_n / \omega \rceil$
- Then ($||$ indicates concatenation, $0 \leq a_i < 2^\omega$)

$$a = a_{\kappa-1} || a_{\kappa-2} || \dots || a_0,$$

- Note that some a_i might be 0 if the bit length of a is less than ℓ_n. We have

$$a = \sum_{i=0}^{\kappa-1} a_i (2^\omega)^i.$$

Example

$a = 55 = 110111_2$, $n = 69$, $\omega = 2$.

$$\ell_n = ? \quad \kappa = \lceil \ell_n / \omega \rceil = ?$$

Notations – Example

- n: an integer of bit length ℓ_n
- $a, b \in \mathbb{Z}_n$, in particular, $0 \le a, b < n$.
- Let $\kappa = \lceil \ell_n / \omega \rceil$
- Then ($||$ indicates concatenation, $0 \le a_i < 2^\omega$)

$$a = a_{\kappa-1} || a_{\kappa-2} || \ldots || a_0,$$

- Note that some a_i might be 0 if the bit length of a is less than ℓ_n. We have

$$a = \sum_{i=0}^{\kappa-1} a_i (2^\omega)^i.$$

Example

$a = 55 = 110111_2$, $n = 69$, and $\omega = 2$.

$$\ell_n = 7, \quad \kappa = \lceil \ell_n / \omega \rceil = \lceil 7/2 \rceil = 4.$$

$$a_0 = ? \quad a_1 = ? \quad a_2 = ? \quad a_3 = ?$$

Notations – Example

- n: an integer of bit length ℓ_n, i.e.
- $a, b \in \mathbb{Z}_n$, in particular, $0 \leq a, b < n$.
- Let $\kappa = \lceil \ell_n / \omega \rceil$
- Then ($||$ indicates concatenation, $0 \leq a_i < 2^\omega$)

$$a = a_{\kappa-1} || a_{\kappa-2} || \dots || a_0,$$

- Note that some a_i might be 0 if the bit length of a is less than ℓ_n. We have

$$a = \sum_{i=0}^{\kappa-1} a_i (2^\omega)^i.$$

Example

$a = 55 = 110111_2$, $n = 69$, $\omega = 2$. $\ell_n = 7$, $\kappa = \lceil \ell_n/\omega \rceil = \lceil 7/2 \rceil = 4$.

$$a_0 = 11 = 3, \quad a_1 = 01 = 1, \quad a_2 = 11 = 3, \quad a_3 = 0.$$

$$a = 3 \times (2^2)^0 + 1 \times (2^2)^1 + 3 \times (2^2)^2 + 0 \times (2^2)^3 = 3 + 4 + 48 + 0 = 55.$$

Blakley's method

- We would like to compute

$$R := ab \bmod n, \quad a, b \in \mathbb{Z}_n.$$

- We have discussed that

$$a = \sum_{i=0}^{\kappa-1} a_i (2^{\omega})^i,$$

where $0 \leq a_i < 2^{\omega}$.

- The product ab can be computed as follows

$$t = ab = \left(\sum_{i=0}^{\kappa-1} a_i (2^{\omega})^i \right) b = \sum_{i=0}^{\kappa-1} (2^{\omega})^i a_i b,$$

Algorithm 3: Blakely's method for computing modular multiplication.

Input: n, a, b// $n \in \mathbb{Z}, n \geq 2$ has bit length ℓ_n; $a, b \in \mathbb{Z}_n$

Output: $R = ab \bmod n$

$R = 0$

// $\kappa = \lceil \ell_n / \omega \rceil$, where ω is the word size of the computer

for $i = \kappa - 1$, $i \geq 0$, $i - -$ **do**

  $R = 2^{\omega} R + a_i b$

  $R = R \bmod n$

return R

Blakely's method

Input: $n, \ a, \ b$
Output: $R = ab \bmod n$

```
1 R = 0
2 for i = κ − 1, i ≥ 0, i − − do
3     R = 2^ω R + a_i b
4     R = R mod n
5 return R
```

Line 3,

$$R \leq 2^{\omega}(n-1)+(2^{\omega}-1)(n-1) = (2^{\omega+1}-1)n-(2^{\omega+1}-1)$$

Line 4 can be replaced by comparing R with n for $2^{\omega+1}-2$ times and subtract n from R in case $R \geq n$:

```
for j = 0, 1, 2 ..., 2^(ω+1) − 2 do
    if R ≥ n then  R = R − n
    else break
```

in this way, we can avoid dividing by n

Blakely's method – Example

Input: $n, \ a, \ b$
Output: $R = ab \bmod n$

$R = 0$
for $i = \kappa - 1, \ i \geq 0, \ i - -$ **do**
    $R = 2^{\omega} R + a_i b$
    $R = R \bmod n$
return R

Example

$\omega = 2$, $a = 13 = 1101_2$, $b = 5$, $n = 15$ $(\ell_n = 4)$, $\kappa = 2$.

$$a_0 = 01_2 = 1, \quad a_1 = 11_2 = 3.$$

For $i = 1$,

$$R = 0 + 3 \times 5 \bmod 15 = 0 \bmod 15.$$

For $i = 0$,

$$R = ?$$

Blakely's method – Example

Input: $n,\ a,\ b$
Output: $R = ab \bmod n$

```
R = 0
for i = κ − 1, i ≥ 0, i − − do
    R = 2^ω R + a_i b
    R = R mod n
return R
```

Example

$\omega = 2$, $a = 13 = 1101_2$, $b = 5$, $n = 15$ $(\ell_n = 4)$, $\kappa = 2$.

$$a_0 = 01_2 = 1, \quad a_1 = 11_2 = 3.$$

For $i = 1$,

$$R = 0 + 3 \times 5 \bmod 15 = 0 \bmod 15.$$

For $i = 0$,

$$R = 0 + 1 \times 5 \bmod 15 = 5 \bmod 15$$

We have the final result $13 \times 5 \bmod 15 = 5$.

Blakely's method – Example

Input: $n,\ a,\ b$
Output: $R = ab \bmod n$

```
1 R = 0
2 for i = κ − 1, i ≥ 0, i − − do
3     R = 2^ω R + a_i b
4     R = R mod n
5 return R
```

Example

$a = 55 = 110111_2$, $b = 46$, $n = 69$, $\omega = 2$, $\ell_n = 7$, $\kappa = 4$, $a_0 = 11 = 3$, $a_1 = 01 = 1$, $a_2 = 11 = 3$, $a_3 = 0$

$i = 3$ line 3, $R =?$

line 4, $R =?$

Blakely's method – Example

Input: $n,\ a,\ b$
Output: $R = ab \bmod n$

1 $R = 0$
2 **for** $i = \kappa - 1,\ i \geq 0,\ i - -$ **do**
3 $\quad R = 2^{\omega} R + a_i b$
4 $\quad R = R \bmod n$
5 **return** R

Example

$a = 55 = 110111_2$, $b = 46$, $n = 69$, $\omega = 2$, $\ell_n = 7$, $\kappa = 4$, $a_0 = 11 = 3$, $a_1 = 01 = 1$, $a_2 = 11 = 3$, $a_3 = 0$

$i = 3$	line 3,	$R = 0$,
	line 4,	$R = 0$,
$i = 2$	line 3,	$R = ?$
	line 4,	$R = ?$

Blakely's method – Example

Input: $n,\ a,\ b$
Output: $R = ab \bmod n$

```
1  R = 0
2  for i = κ − 1, i ≥ 0, i − − do
3      R = 2^ω R + a_i b
4      R = R mod n
5  return R
```

Example

$a = 55 = 110111_2$, $b = 46$, $n = 69$, $\omega = 2$, $\ell_n = 7$, $\kappa = 4$, $a_0 = 11 = 3$, $a_1 = 01 = 1$, $a_2 = 11 = 3$, $a_3 = 0$

$i = 3$	line 3,	$R = 0,$
	line 4,	$R = 0,$
$i = 2$	line 3,	$R = 3 \times 46 = 138,$
	line 4,	$R = 138 \bmod 69 = 0,$
$i = 1$	line 3,	$R =?$
	line 4,	$R =?$
$i = 0$	line 3,	$R =?$
	line 4,	$R =?$

Blakely's method – Example

```
   Input: n, a, b
   Output: R = ab mod n
1  R = 0
2  for i = κ − 1, i ≥ 0, i − − do
3      R = 2^ω R + a_i b
4      R = R mod n
5  return R
```

Example

$a = 55 = 110111_2$, $b = 46$, $n = 69$, $\omega = 2$, $\ell_n = 7$, $\kappa = 4$, $a_0 = 11 = 3$, $a_1 = 01 = 1$, $a_2 = 11 = 3$, $a_3 = 0$

$i = 3$	line 3,	$R = 0$,
	line 4,	$R = 0$,
$i = 2$	line 3,	$R = 3 \times 46 = 138$,
	line 4,	$R = 138 \bmod 69 = 0$,
$i = 1$	line 3,	$R = 1 \times 46 = 46$,
	line 4,	$R = 46 \bmod 69 = 46$,
$i = 0$	line 3,	$R = 2^2 \times 46 + 3 \times 46 = 322$,
	line 4,	$R = 322 \bmod 69 = 46$.

Final remarks

- Currently, a few hundred qubits (a quantum counterpart to the classical bit) are possible for a quantum computer
- To break RSA, thousands of qubits are required.
- Post-quantum public key cryptosystems are being proposed to protect communications after a sufficiently strong quantum computer is built.
- Various public key cryptosystems based on different problems
- Various digital signature designs
- Provable secure signature, similarity to one-time pad