CISC 468: CRYPTOGRAPHY

LESSON 10: THE RSA CRYPTOSYSTEM, CONTINUED

Furkan Alaca

READINGS

- Section 7.4: Encryption and Decryption: Fast Exponentiation, Paar & Pelzl
- Section 7.7: RSA in Practice: Padding, Paar & Pelzl
- Section 7.8: Attacks, Paar & Pelzl

EXPONENTIATION IN RSA

- RSA and other public-key algorithms rely on arithmetic with very large numbers
- Recall the RSA encryption and decryption functions:

$$y = e_{k_{pub}}(x) = x^e \mod n,$$

$$x = d_{k_{pr}}(y) = y^d \mod n.$$

- The exponents e and d are very large (2048 bits or more)

EXPONENTIATION: SIMPLE METHOD

$$x \xrightarrow{SQ} x^2 \xrightarrow{MUL} x^3 \xrightarrow{MUL} x^4 \xrightarrow{MUL} x^5 \dots$$

- Naive approach: If the exponent is $\sim 2^{2048}$, then the base would need to be multiplied by itself $\sim 2^{2048}$ times

EXPONENTIATION: A FASTER METHOD

Computing x^8 with the simple method (7 multiplications):

$$x \xrightarrow{SQ} x^2 \xrightarrow{MUL} x^3 \xrightarrow{MUL} x^4 \xrightarrow{MUL} x^5 \xrightarrow{MUL} x^6 \xrightarrow{MUL} x^7 \xrightarrow{MUL} x^8$$

Computing x^8 by squaring three times (3 multiplications):

$$x \xrightarrow{SQ} x^2 \xrightarrow{SQ} x^4 \xrightarrow{SQ} x^8$$

But this method only works if the exponent is a power of 2 — can we extend it to work with arbitrary exponents?

EXPONENTIATION: FASTER METHOD FOR ARBITRARY EXPONENTS

We can compute x^{26} as follows, with 6 multiplications (vs. 25 with the simple method):

$$x \xrightarrow{SQ} x^2 \xrightarrow{MUL} x^3 \xrightarrow{SQ} x^6 \xrightarrow{SQ} x^{12} \xrightarrow{MUL} x^{13} \xrightarrow{SQ} x^{26}$$

- In this example we know the sequence of squaring and multiplying that needs to be done to end up with x^{26}
- To determine the sequence for any exponent, we can use the *square-and-multiply* algorithm

SQUARE-AND-MULTIPLY ALGORITHM

1. To square a number x^a, first write the binary representation of the exponent a.
2. Iterate from the most-significant bit to the least-significant bit of a.
    1. To process the most-significant bit, write down x.
    2. For each remaining bit: Square the result from the previous iteration. If the current bit is a 1, follow up the squaring with a multiplication by x.

SQUARE-AND-MULTIPLY ALGORITHM: EXAMPLE

Consider the exponentiation x^{26}.

1. The binary representation of 26 is 11010.
2. Proceed with the square-and-multiply algorithm as follows:

SQUARE-AND-MULTIPLY ALGORITHM: INTUITION

- In binary, squaring a number involves shifting the exponent to the left (i.e., appending a 0)
- Multiplying by x results in adding 1 to the exponent

Current bit	Result	Result (w/ exponent in binary)
1	x	x^{1}
1	$(x)^2 \cdot x = x^3$	x^{11}
0	$(x^3)^2 = x^6$	x^{110}
1	$(x^6)^2 \cdot x = x^{13}$	x^{1101}
0	$(x^{13})^2 = x^{26}$	x^{11010}

SQUARE-AND-MULTIPLY ALGORITHM: COMPLEXITY

- For an exponent of bit length $t + 1$:
  - The number of required square operations is t
  - The number of multiplications required is $0.5t$ on average
- For a 1024-bit exponent:
  - Simple exponentiation would take 2^{1024} multiplications (which is computationally infeasible)
  - The square-and-multiply algorithm would require $1.5 \times 1024 = 1536$ multiplications

SCHOOLBOOK RSA

We have so far seen "schoolbook RSA", which has weaknesses:

- It is deterministic: For a specific key, a particular plaintext is always mapped to the same ciphertext
- Attacker can derive statistical properties of the plaintext from the ciphertext
- Given some plaintext-ciphertext pairs, partial information can be derived from new ciphertexts encrypted with the same key
- Plaintexts $x = 0, x = 1, x = -1$ produce the ciphertexts $y = 0, y = 1, y = -1$, respectively
- Small public exponents e and small plaintexts x may be subject to attack

MALLEABILITY OF RSA

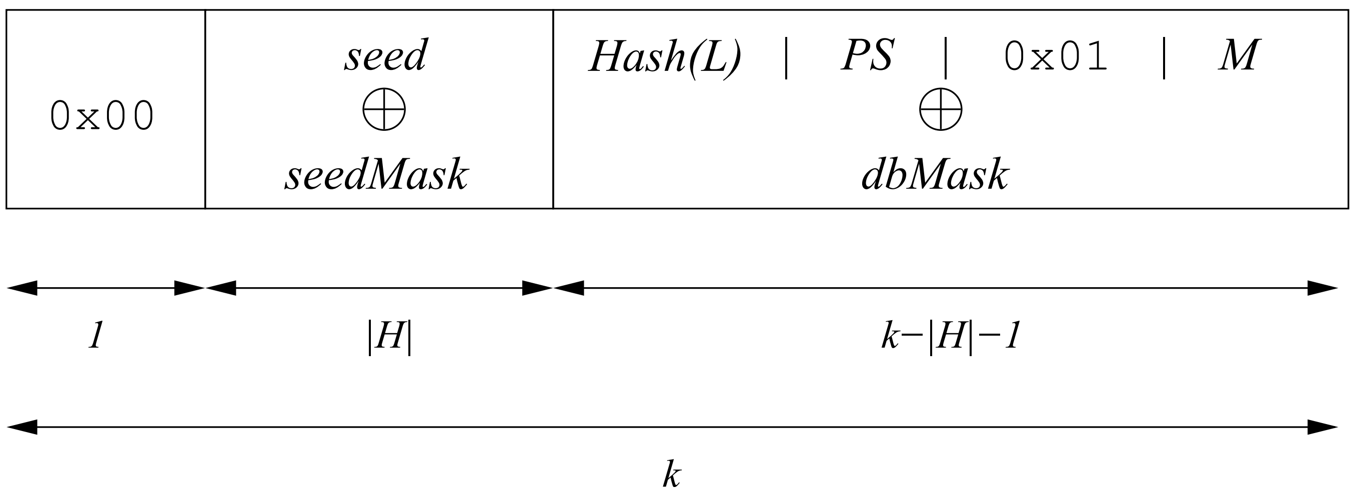
- RSA is also *malleable*: Even if the attacker is not able to decrypt the ciphertext, they can manipulate it
- Example: Consider a financial transaction where the ciphertext y is the amount of money to be sent
  - An attacker can replace y with $2^e y$, where e is the public exponent
  - This will then decrypt to $2x$, which is double the amount of money that should have been sent

PADDING

- Schoolbook RSA's weaknesses can be mitigated with a properly-implemented padding scheme
- Padding embeds randomness into the plaintext before encryption
- A common method defined in the PKCS #1 standard is Optimal Asymmetric Encryption Padding (OAEP)

PADDING: OAEP

- M is the plaintext
- k is the length of the modulus, which is the maximum length that a single RSA encryption operation can encrypt
- `seed` is a randomly-generated value of length $|H|$ bytes
- `seedMask` and `dbMask` are generated using a mask generation function, which internally uses a *hash function*,



ATTACKS ON RSA

- The numerous attacks against RSA proposed since 1977 typically exploit weaknesses in RSA implementations, rather than the algorithm itself
- There are three general attack families against RSA:
  - Protocol attacks
  - Mathematical attacks
  - Side-channel attacks

RSA: PROTOCOL ATTACKS

- Protocol attacks exploit the malleability of RSA
- Such attacks can be avoided with proper use of padding
- To avoid these attacks, strictly follow modern security standards that describe exactly how RSA should be used

RSA: MATHEMATICAL ATTACKS

- The best known mathematical attack is to factor the modulus
- Attacker knows the modulus n, public key e, and ciphertext y
  - They should not know $\Phi(n)$ or the private exponent e
- If n can be factored to obtain p and q, the ciphertext can be recovered in three steps:

$$\Phi(n) = (p - 1)(q - 1)$$

$$d^{-1} \equiv e \bmod \Phi(n)$$

$$x \equiv y^d \bmod n$$

RSA: MATHEMATICAL ATTACKS (2)

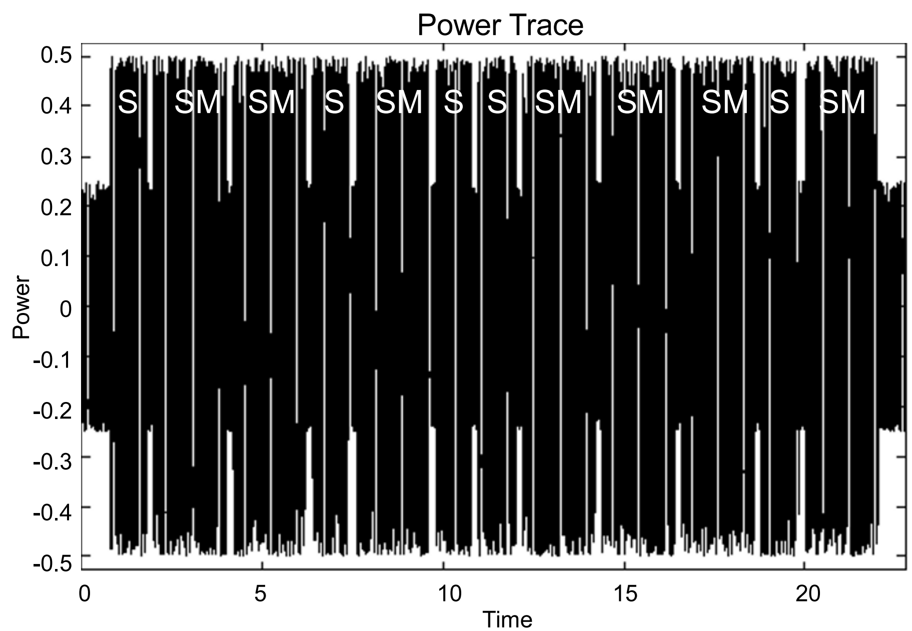
- Significant advances in integer factorization have been made over the last few decades
- This progress is largely due to improved factorization algorithms developed due to interest sparked by RSA
  - And to a lesser extent due to improved computer technology
- In the 1990s, a 1024-bit modulus was recommended for RSA, but this was gradually phased out and is currently disallowed by the latest guidance by NIST and others
  - The current recommendation is to use at least 2048-bit

RSA: SIDE-CHANNEL ATTACKS

- Side-channel attacks exploit information about the private key leaked through physical channels such as timing or power consumption
  - Typically requires fine-grained measurements, which often (but not always) necessitates physical access to the device

RSA SIDE-CHANNEL ATTACKS: SIMPLE POWER ANALYSIS (SPA)

- Microprocessor power trace below reveals the square-and-multiply sequence, revealing private exponent d
  - Short power spike indicates square operation (0 bit)
  - Long power spike indicates square-and-multiply (1 bit)



RSA SIDE-CHANNEL ATTACKS: DEFENSES

- A simple countermeasure against SPA is to execute a "dummy" multiplication when iterating over the 0 bits
  - Ensures that a square-and-multiply takes place for each bit
- Defenses against more advanced side-channel attacks are not always as straightforward
- Remember: **Don't roll your own crypto**

ON-PATH ATTACKS: PASSIVE VS. ACTIVE

- Our biggest challenge with symmetric cryptography was communicating the secret key to be used between the two parties
  - This challenge does not exist with RSA, since public keys can be freely communicated
- But we considered only *passive attacks*, where the attacker can **read** the data exchanged between sender and receiver
- *Active attacks*, where the attacker can **modify** the data, are more powerful and will be discussed later
  - Consider the implications when retrieving public keys