Cryptography

Public Key Cryptography & RSA

M S Vilkhu

-xx08 Nov 2024(C5) xx

04 Nov 2024(C1/C3)

02 Nov 2024(C1/C3/C5)

16 Nov 2024(C1/C3/C5)

The RSA Algorithm

Topics

Topics covered

- **Principles** of Public-Key Cryptosystems
- Public-Key Cryptosystems
- **Applications** for Public-Key Cryptosystems
- **Requirements** for Public-Key Cryptography
- Public-Key **Cryptanalysis**

This session

- **The RSA Algorithm**
- **Description of the Algorithm**
- **Computational Aspects**
- **The Security of RSA**

The RSA Algorithm

- The **pioneering paper** by Diffie and Hellman [DIFF 76b] introduced a **new approach to cryptography**
- challenged **cryptologists** to come up with a **crypto graphic algorithm** that met the requirements for public-key systems.
- A number of algorithms have been proposed for public-key cryptography. Some of these, though **initially promising**, turned out to be **breakable.**
- One of the **first successful** responses to the challenge was developed in 1977 by **Ron Rivest, Adi Shamir, and Len Adleman** at **MIT** and first published in 1978 [RIVE78].
- The **Rivest-Shamir-Adleman (RSA)** scheme has since that time **reigned supreme** as the most **widely accepted and implemented** general-purpose approach to public-key encryption.

The RSA Algorithm

- The RSA scheme is a cipher in which the **plaintext and ciphertext are integers between 0 and n — 1 for some n**.
- A typical size for **n is 1024 bits**, or **309 decimal digits**.
- That is, **n is less than 2^{1024}**.

- examine RSA with an **explanation** of the algorithm.
- Then examine some of the **computational** and **cryptanalytical implications** of RSA.

Description of RSA Algorithm

- RSA makes use of an expression with exponentials.
- Plaintext is **encrypted in blocks**, with each **block** having a
  - binary value **less than** some **number n.**
  - the **block size** must be less than or equal to $\log_2(n) + 1$;
  - in practice, the block size is **i bits**, where $2^i < n\ 2^{i+l}$.
- Encryption and decryption are of the following form, for some **plaintext block M** and **ciphertext block C**.

$$C = M^e \bmod n$$

$$M = C^d \bmod n$$

$$= (M^e)^d \bmod n$$

$$= M^{ed} \bmod n$$

Description of RSA Algorithm

- Both sender and receiver must know the **value of n.**
- The **sender knows** the value of **e**, and
- only the **receiver knows** the value of **d.**
- Thus, this is a public- key encryption algorithm with
  - **public key** of PU = {e, n}
  - **private key** of PR = {d, n}.

Description of RSA Algorithm

For this algorithm to be satisfactory for public-key encryption, the **following requirements must be met.**

1. It is possible to find values of ***e, d,* and *n*** such that $M^{ed} \bmod n = M$ for all $M < n$.

2. It is relatively easy to calculate $M^{e} \bmod n$ and $C^{d} \bmod n$ for all values of $M < n$.

3. It is infeasible to determine ***d*** given ***e* and *n***.

Description of RSA Algorithm

- Focus on **first requirement** and find

$$M^{ed} \bmod n = M$$

- The preceding relationship holds **if *e* and *d* are multiplicative inverses modulo** $\emptyset(n)$,
- where $\emptyset(n)$, is the **Euler totient function**,
- for *p, q* prime,

$\emptyset(pq) = (p - 1)(q - 1)$

- The relationship between *e* and *d* can be expressed as

$$ed \bmod \emptyset(n) = 1$$

- This is equivalent to saying

$$ed \equiv 1 \bmod \emptyset(n)$$

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

That is, *e* and *d* are multiplicative inverses mod $\emptyset(n)$.

Note that, according to the rules of modular arithmetic, this is true only if *d* (and therefore *e*) is **relatively prime** to $\emptyset(n)$. Equivalently, $\gcd(\emptyset(n), d) = 1$.

RSA scheme

- **RSA scheme – the ingredient are**

p, q, two prime numbers (private, chosen)

$\boldsymbol{n} = pq$ (public, calculated)

$\boldsymbol{e}$, with $\gcd(\emptyset(n), e) = 1$; $1 < e < \emptyset(n)$ (public, chosen)

$\boldsymbol{d} \equiv e^{-1} \pmod{\emptyset(n)}$ (private, calculated)

- The **private key consists of {d, n} and the public key consists of {e, n}.**
- Suppose that **user A** has **published its public key** and that **user B wishes** to send the message M to A.
- Then B calculates $\boldsymbol{C = M^e \bmod n}$ and transmits C.
- On receipt of this ciphertext, user A decrypts by calculating

$M = C^d \bmod n$.

RSA scheme

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Public Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

RSA Example

- **Key generation**

1. Select **two prime numbers**, $p = 17$ and $q = 11$.

2. Calculate $n = pq = 17 * 11 = 187$.

3. Calculate $\emptyset(n) = (p - 1)(q - 1) = 16 * 10 = 160$.

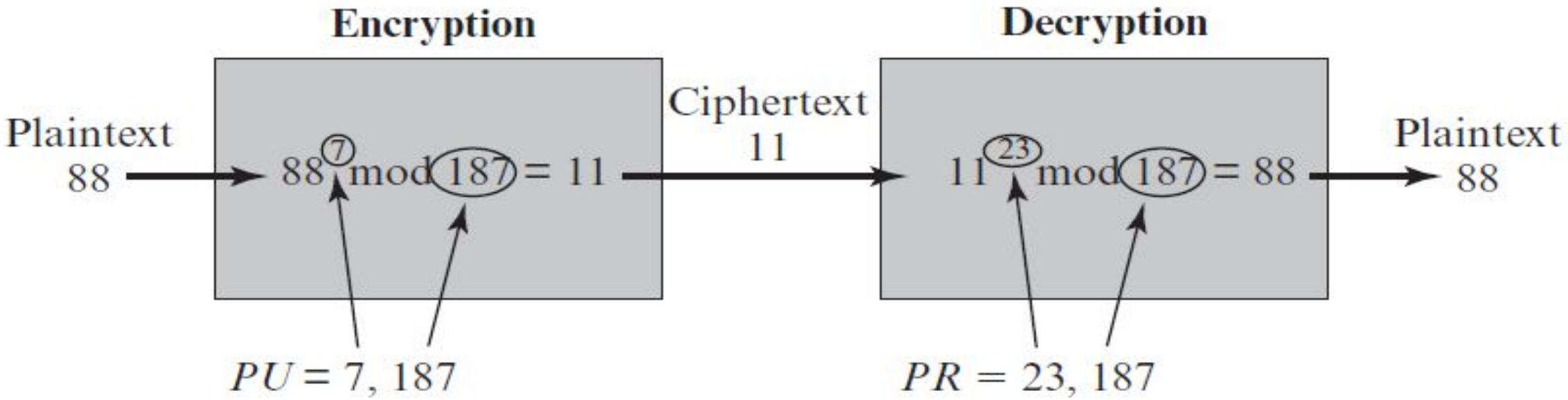
4. Select e such that e is relatively prime to $\emptyset(n) = 160$ and less than $\emptyset(n)$; we choose $e = 7$.

5. Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$.

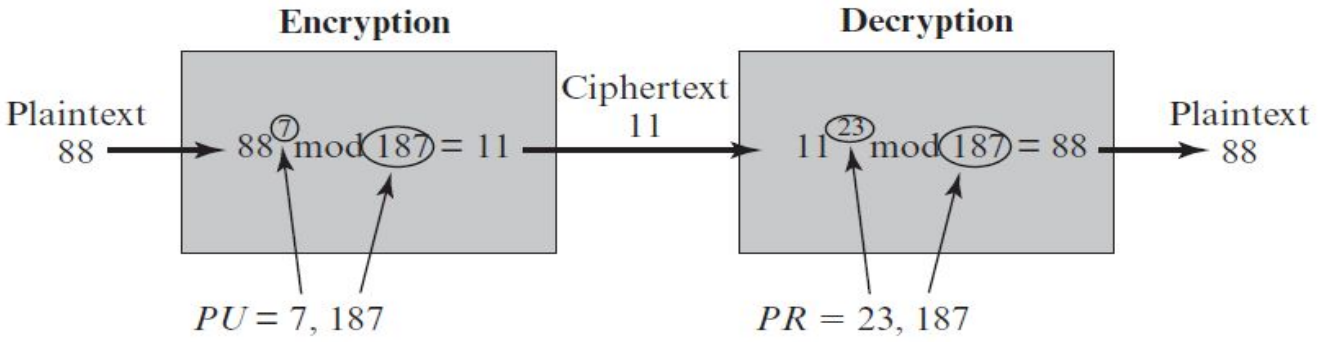
The correct value is $d = 23$, because $23 * 7 = 161 = (1 * 160) + 1$; d can be calculated using the extended Euclid's algorithm

The resulting keys are public key **PU = {7, 187}** and private key **PR = {23, 187}.**

RSA Example


Encryption
Decryption
Plaintext
88
$88^{7} \bmod 187 = 11$
Ciphertext
11
$11^{23} \bmod 187 = 88$
Plaintext
88
$PU = 7, 187$
$PR = 23, 187$

RSA Example



- The example shows the use of these keys for a plaintext input of **M = 88**. For encryption, we need to calculate C = 88^7 mod 187.
- Exploiting the properties of modular arithmetic, we can do this as follows.

$88^7 \bmod 187 = [(88^4 \bmod 187) * (88^2 \bmod 187) * (88^1 \bmod 187)] \bmod 187$

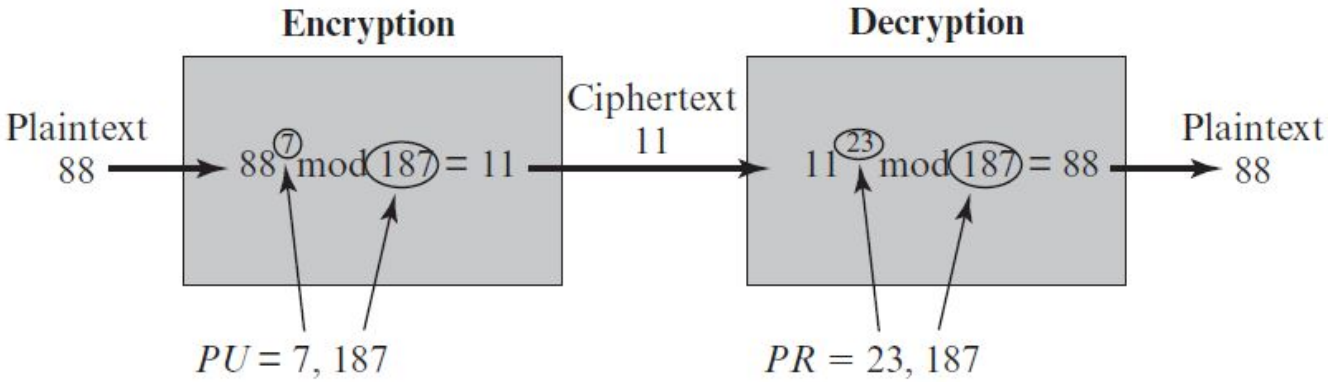
$88^1 \bmod 187 = 88$

$88^2 \bmod 187 = 7744 \bmod 187 = 77$

$88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$

$88^7 \bmod 187 = (88 * 77 * 132) \bmod 187 = 894{,}432 \bmod 187 = 11$

RSA Example



- For decryption, we calculate M = 11^{23} mod 187:

$11^{23} \bmod 187 = [(11^{1} \bmod 187) * (11^{2} \bmod 187) * (11^{4} \bmod 187) * (11^{8} \bmod 187) * (11^{8} \bmod 187)] \bmod 187$

$11^{1} \bmod 187 = 11$

$11^{2} \bmod 187 = 121$

$11^{4} \bmod 187 = 14{,}641 \bmod 187 = 55$

$11^{8} \bmod 187 = 214{,}358{,}881 \bmod 187 = 33$

$11^{23} \bmod 187 = (11 * 121 * 55 * 33 * 33) \bmod 187$

$= 79{,}720{,}245 \bmod 187 = 88$

Computational Aspects

Computational Aspects

- We now turn to the **issue of the complexity of the computation** required to use RSA. There are actually **two issues to consider**:

1. **encryption/decryption and**
2. **key generation.**

- first at the process of encryption and decryption and then consider key generation.

Computational Aspects

- **EXPONENTIA TION IN MODULAR ARITHMETIC**
- Both encryption and decryption in RSA involve **raising an integer** to an **integer power**, **mod n**.
- If the exponentiation is **done over the integers** and then **reduced modulo n,** the **intermediate values** would be very large.
- we can make **use of a property of modular arithmetic**:

  [(a mod n) X (b mod n)] mod n = (a X b) mod n

- Thus, **reduce intermediate results modulo n**. This makes the **calculation practical**.
- **Another consideration** is the **efficiency of exponentiation**, because with RSA, we are dealing with potentially large exponents.
- To see how efficiency might be in x^{16} creased, consider that we wish to compute x .
- A straightforward approach requires 15 multiplications:
- x^{16} = x X x X x X x X x X x X x X x X x X x X x X x X x X x
- we **can achieve the same** final result with **only four multiplications** if we **repeatedly** take the **square of each partial result**, successively forming (x^2, x^4, x^8, x^{16}).

Computational Aspects

- **EXPONENTIATION IN MODULAR ARITHMETIC**
- As **another example**, suppose we wish to calculate $\mathbf{x^{11}}$ **mod n** for some integers x and n. Observe that $x^{11} = x^{1+2+8} = (x)(x^2)(x^8)$.
- In this case, we compute
- $x \bmod n$, $x^2 \bmod n$, $x^4 \bmod n$, and $x^8 \bmod n$ and
- then calculate $[(x \bmod n) * (x^2 \bmod n) * (x^8 \bmod n)] \bmod n$.

- We can therefore **develop the algorithm** for computing $\mathbf{a^b}$ **mod n,**

Computational Aspects

- **EFFICIENT OPERATION USING THE PUBLIC KEY**
- To **speed up the operation** of the RSA algorithm using the public key, a **specific choice of e is usually made.**
- The most **common choice is 65537 ($2^{16} + 1$)**; two **other popular choices** are 3 and 17.
- Each of these choices **has only two 1 bits**,
- so the **number of multiplications** required to perform exponentiation is **minimized**.

Computational Aspects

- **EFFICIENT OPERATION USING THE PUBLIC KEY**
- with a **very small public key**, such as e = **3,** RSA becomes **vulnerable to a simple attack**.
- Suppose we have three different RSA users who all **use 3** but have **unique values of n**, namely (n1, n2, n3).
- If user A sends the value **e** the same encrypted message M to all three users, then the three ciphertexts are

  Cl = M^3mod n1, C2 = M^3mod n1, and C3 = M^3mod n3.
- It is likely that n1, n2 and n3 are pairwise relatively prime. Therefore, one can use the **Chinese remainder theorem (CRT)** to compute M^3 mod (n1n2n3).
- By the rules of the RSA algorithm, **M is less than each of the ni**, therefore **M^3< n1n2n3** Accordingly, the attacker need only **compute the cube root of** M^3.
- This attack can be countered by adding a **unique pseudorandom bit string as padding** to each instance of M to be encrypted.

Computational Aspects

- **EFFICIENT OPERATION USING THE PRIVATE KEY**
- We **cannot** similarly **choose a small constant value of d for efficient operation**.
- A **small value** of **d** **is vulnerable to a brute-force attack** and to other forms of cryptanalysis [WIEN90].
- However, there is a **way to speed up computation** using the **Chinese remainder theorem(CRT).**
- We wish to compute the value

$M = C^d \bmod n$.

- Let us define the following intermediate results:

$Vp = C^d \bmod p \quad Vq = C^d \bmod q$

- Apply **Chinese remainder theorem (CRT)** can solve this. (not going into detailed calculations for sake simplicity and keeping focus on concepts)

Computational Aspects

- **KEY GENERATION**
- Before the application of the public-key cryptosystem, each participant **must generate a pair of keys.** This involves the following tasks.
  - Determining **two prime numbers**, p and q.
  - Selecting either **e or d** and **calculating** the other.
- **First, selection of p and q**.

Computational Aspects

- KEY GENERATION
- **First, selection of p and q**.
- Because the value of **n = pq** will be **known to any potential adversary**, in order to prevent the discovery of p and q by exhaustive methods, **these primes must be chosen from a sufficiently large set** (i.e., **p and q must be large numbers**).
- On the other hand, the method used for **finding large primes must be reasonably efficient**.
- At present, there are **no useful techniques** that yield **arbitrarily large primes**, so some other means of tackling the problem is needed.
- The **procedure** that is generally used is to **pick at random an odd number** of the desired order of **magnitude and test whether that number is prime**. If **not, pick successive random numbers until one is found that tests prime.**

Computational Aspects

- **KEY GENERATION**
- **First, selection of p and q**.
- A **variety of tests** for primality have been developed (e.g., see [KNUT98] for a description of a number of such tests).
- the **tests are probabilistic**. That is, the test **will merely determine that a given integer is probably prime.**
- Despite **this lack of certainty**, these tests **can be run in such a way as to make the probability as close to 1.0 as desired**.
- As an example, one of the more efficient and **popular algorithms, the Miller—Rabin algorithm**.
- With this algorithm and most such algorithms, the **procedure for testing whether a given integer n is prime is to perform** some calculation that involves n and a randomly chosen integer a.
- If **n "fails" the test**, then **n is not prime.**
- If **n "passes" the test, then n may be prime or nonprime**. If n passes many such tests with **many different randomly chosen values for a,** then we can have high **confidence** that n is, in fact, **prime**.

Computational Aspects

- **KEY GENERATION**
- **First, selection of p and q**.

In summary, the procedure for picking a prime number is as follows.

1. **Pick an odd integer** n at random (e.g., using a pseudorandom number generator).
2. Pick an **integer a < n at random**.
3. Perform the **probabilistic** primality test, such as **Miller—Rabin**, with a as a parameter. If n fails the test, reject the value n and go to step 1.
4. If **n has passed a sufficient number** of tests, accept n; otherwise, go to step 2.

Computational Aspects

- **KEY GENERATION**
- **First, selection of p and q**.
- Having **determined prime numbers p and q**,
- the process **of key generation is completed** by selecting a **value of e** and **calculating d** or, alternatively, **selecting a value of d and calculating e**.
- select an **e such that gcd(Ø(n), e) = 1**, then calculate **$d = e^{-1} \pmod{Ø(n)}$**
- determine the **inverse** of one of the integers modulo the other. The algorithm, referred to as the extended **Euclid's algorithm**,
- Thus, the **procedure is to generate a series of random numbers**, **testing** each against Ø(n) until a number **relatively prime** to Ø(n) is found.

The security of RSA

- Brute force:
- Mathematical attacks:.
- Timing attacks:
- Hardware fault-based attack:
- Chosen ciphertext attacks:

The security of RSA

Five possible approaches to attacking the RSA algorithm are

- **Brute force:** This involves trying **all possible** private keys.
- **Mathematical attacks**: There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks**: These depend on the **running time** of the decryption algorithm.
- **Hardware fault-based attack:** This involves **inducing hardware faults in the processor** that is generating digital signatures.
- **Chosen ciphertext attacks**: This type of attack **exploits properties** of the RSA algorithm.

The security of RSA

■ **Brute force:**
■ Mathematical attacks:.
■ Timing attacks:
■ Hardware fault-based attack:
■ Chosen ciphertext attacks:

- Five possible approaches to attacking the RSA algorithm are

■ **Brute force:** This involves **trying all possible private keys.**

- The **defense** against the brute-force approach is the same for RSA as for other cryptosystems, namely, to **use a large key space**.
- Thus, the **larger the number of bits** in d, the better.
- However, because the **calculations involved**, both in key generation and in encryption/decryption, **are complex**, the larger the size of the key, the **slower** the system will run.

- Brute force:
- **Mathematical attacks:.**
- Timing attacks:
- Hardware fault-based attack:
- Chosen ciphertext attacks:

The Security of RSA

- **Attacking RSA mathematically** - **The factoring problem**
- We can identify **three approaches** to attacking RSA mathematically.

1. **Factor n** into its **two prime factors**. This enables calculation of **Ø (n) = (p - 1) x (q - 1)**, which in turn enables determination of $\mathbf{d = e^{-1} \pmod{Ø (n)}}$.
2. Determine **Ø (n)** directly, **without first determining p and q**. Again, this **enables determination of d = (mod Ø (n)).**
3. Determine **d directly**, without **first determining Ø (n).**

The Security of RSA

- **Attacking RSA mathematically - The factoring problem**

1. **Factor n** into its **two prime factors**.

This enables calculation of **Ø (n) = (p - 1) x (q - 1)**, which in turn enables determination of **d = e^{-1} (mod Ø (n))**.

- cryptanalysis of RSA have **focused** on the **task of factoring n** into its **two prime factors**. Determining Ø(n) given n is equivalent to factoring n [RIBE96].
- With presently **known algorithms**, **determining d given e and n -** appears to be at least as **time-consuming as the factoring problem** [KAL195].

can use **factoring performance** as a **benchmark** against which to evaluate security of RSA

- Brute force:
- **Mathematical attacks:.**
- Timing attacks:
- Hardware fault-based attack:
- Chosen ciphertext attacks:

The Security of RSA

- **Attacking RSA mathematically - The factoring problem**

1. **Factor n** into its **two prime factors**

For **a large n with large prime factors**, **factoring is a hard problem**, but it is **not as hard** as it used to be.

- A striking **illustration** of this is the following.
- In 1977, the **three inventors of RSA dared Scientific American readers** to decode a cipher they printed in Martin Gardner's "**Mathematical Games" column** [GARD77]. **They offered a $100** reward for the return of a **plaintext sentence**, an event they predicted might not **occur for some 40 quadrillion years**.
- In **April of 1994**, a group working over the Internet **claimed the prize after only eight months of work** [LEUT94]. **This challenge** used a **public key size** (length of n) of **129 decimal digits**, or **around 428 bits**.
- In the meantime, just as they had done for DES, RSA Laboratories had issued challenges for the **RSA cipher with key sizes of 100, 110, 120, and so on, digits**.

- Brute force:
- **Mathematical attacks:.**
- Timing attacks:
- Hardware fault-based attack:
- Chosen ciphertext attacks:

The Security of RSA

- **Attacking RSA mathematically - The factoring problem**

1. Factor n into its **two prime factors**

- The latest challenge to be met is the RSA-768 challenge with a key length of 232 decimal digits, or 768 bits. Table shows the results.

Table: Progress in factorization

Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

The Security of RSA

- Brute force:
- **Mathematical attacks:.**
- Timing attacks:
- Hardware fault-based attack:
- Chosen ciphertext attacks:

- **Attacking RSA mathematically - The factoring problem**

1. Factor n into its **two prime factors**

- A **striking fact** about the progress reflected in Table concerns the **method used**.
- Until the **mid-1990s,** factoring attacks were made using an approach known as the **quadratic sieve.**
- The attack on **RSA-130** used a **newer algorithm**, the **generalized number field sieve (GNFS),** and was able to **factor a larger number** than RSA-129 at **only 20% of the computing effort.**

Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

Table – Progress in factorization

- Brute force:
- Mathematical attacks:.
- Timing attacks:
- Hardware fault-based attack:
- Chosen ciphertext attacks:

The Security of RSA

- **Attacking RSA mathematically - The factoring problem**

1. Factor n into its **two prime factors**

- **The threat to larger key sizes is twofold**:
  - the continuing **increase in computing power** and
  - the continuing **refinement of factoring algorithms**.
- **Observation** - that the move to a **different algorithm** resulted in a **tremendous speedup**.
- We can **expect further refinements in the generalized number field sieve (GNFS)**, and the use of an **even better algorithm** is also a possibility. In fact, a related algorithm, the **special number field sieve (SNFS),**

The Security of RSA

- Brute force:
- Mathematical attacks:.
- Timing attacks:
- Hardware fault-based attack:
- Chosen ciphertext attacks:

- **Attacking RSA mathematically - The factoring problem**

1. Factor n into its **two prime factors**

- Thus, we **need to be careful in choosing a key size for RSA**.
- The team that **produced the 768-bit factorization** [KLEIIO] observed that factoring a **1024-bit RSA modulus would be about a thousand times harder** than factoring a 768-bit modulus, and a **768-bit RSA modulus is several thousands times** harder to factor than a **512-bit** one.
- Based on the **amount of time between** the **512-bit and 768-bit** factorization successes, the team felt it to be reasonable to expect that the **1024-bit RSA moduli could be factored well within the next decade** by a similar academic effort. Thus, they **recommended phasing** out usage of **1024-bit RSA** within the next few years (from 2010).

The Security of RSA

- Brute force:
- Mathematical attacks:.
- Timing attacks:
- Hardware fault-based attack:
- Chosen ciphertext attacks:

- **Attacking RSA mathematically - The factoring problem**

1. Factor n into its **two prime factors**

- In addition to **specifying the size of n,** a number of **other constraints** have been suggested by **researchers**. To **avoid values of n** that may be factored more easily, the algorithm's inventors suggest the following **constraints on p and q**.

1. **p and q should differ in length <u>by only a few digits</u>**. Thus, for a 1024-bit key (309 decimal digits), both p and q should be on the order of **magnitude of 10^{75} to 10^{100}**.

2. Both (p - 1) and (q - 1) should contain a **large prime factor**.

3. gcd(p - 1, q - 1) should be **small**.

gcd stands for **greatest common divisor**

The security of RSA

- Brute force:
- Mathematical attacks:.
- **Timing attacks:**
- Hardware fault-based attack:
- Chosen ciphertext attacks:

Timing attacks: These depend on the **running time** of the decryption algorithm.

- If one needed yet **another lesson** about **how difficult** it is to **assess the security of a cryptographic algorithm,** the **appearance of timing attacks** provides a stunning one.
- Paul Kocher, a cryptographic consultant, demonstrated that a **snooper can determine a private key by keeping track of how long a computer takes to decipher messages** [KOCH96, KAL196b].
- Timing attacks are applicable **not just to RSA**, but to **other public-key cryptography** systems.

The security of RSA

- Brute force:
- Mathematical attacks:.
- Timing attacks:
- Hardware fault-based attack:
- Chosen ciphertext attacks:

Timing attacks: These depend on the **running time** of the decryption algorithm.

- This attack **is alarming for two reasons**:
  - It comes from a **completely unexpected direction**, and
  - it is a **ciphertext- only attack**.
- A timing attack is somewhat **analogous** to a **burglar guessing the combination** of a safe by observing how long it takes for someone to turn the dial from number to number.

The security of RSA

- Brute force:
- Mathematical attacks:.
- Timing attacks:
- Hardware fault-based attack:
- Chosen ciphertext attacks:

Timing attacks: These depend on the **running time** of the decryption algorithm.

- simple **countermeasures** that can be used, including the following.

■ **Constant exponentiation time**: Ensure that all **exponentiations take the same amount** of time before returning a result. This is a **simple fix** but does **degrade performance**.

■ **Random delay**: Better performance could be achieved by **adding a random delay** to the exponentiation algorithm to **confuse** the timing attack. Kocher points out that if defenders don't add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays.

■ **Blinding**: **Multiply** the ciphertext by a **random number before performing exponentiation**. This process **prevents the attacker from knowing what ciphertext bits** are being **processed inside the computer** and therefore **prevents the bit-by-bit analysis** essential to the timing attack.

The security of RSA

- Brute force:
- Mathematical attacks:.
- Timing attacks:
- **Hardware fault-based attack**:
- Chosen ciphertext attacks:

- **FAULT-BASED ATTACK**
- Still **another unorthodox approach** to attacking RSA is reported in [PELLIO].
- The approach is an **attack on a processor** that is **generating RSA digital signatures**.
- The attack **induces faults in the signature** computation by **reducing the power to the processor.**
- The faults cause the software to **produce invalid signatures**, which can then be **analyzed by the attacker to recover the private key**.
- The authors show how such an **analysis can be done** and then demonstrate it by extracting a **1024-bit private RSA key** in approximately **100 hours**, using a commercially available microprocessor.

The security of RSA

- Brute force:
- Mathematical attacks:.
- Timing attacks:
- **Hardware fault-based attack**:
- Chosen ciphertext attacks:

- **FAULT-BASED ATTACK**
- The attack algorithm involves **inducing single-bit errors** and **observing the results**. The details are provided in [PELLIO], which also references other proposed **hardware fault-based attacks against RSA**.
- This attack, while **worthy of consideration**, does **not appear to be a serious threat to RSA**. Reasons:
  - It requires that the **attacker have physical access** to the target machine and that the attacker is able to **directly control the input power** to the processor.
  - Controlling the **input power would for most hardware require more than simply controlling the AC power**, but would also involve the **power supply control hardware on the chip.**

The Security of RSA

- Brute force:
- Mathematical attacks:.
- Timing attacks:
- Hardware fault-based attack:
- **Chosen ciphertext attacks:**

- **The CHOSEN CIPHER TEXT ATTACK AND OPTIMAL ASYMMETRIC ENCRYPTION PADDING**
- The basic RSA algorithm is vulnerable to a **chosen ciphertext attack (CCA).**
- CCA is defined as an attack in which the adversary **chooses a number of ciphertexts** and is then **given the corresponding plaintexts**, decrypted with the **target's private key**.
- Thus, the adversary could **select a plaintext**, encrypt it with the **target's public key**, and then be able to get the **plaintext back by having it decrypted** with the private key.
- no new information. Instead, the adversary **exploits properties of RSA** and **selects blocks of data that**, when processed using the target's private key, yield information needed for **cryptanalysis**.
- overcome this simple attack…

- Brute force:
- Mathematical attacks:.
- Timing attacks:
- Hardware fault-based attack:
- **Chosen ciphertext attacks**:

The Security of RSA

- **The CHOSEN CIPHER TEXT ATTACK AND OPTIMAL ASYMMETRIC ENCRYPTION PADDING**
- To **overcome this simple attack**,
- practical RSA-based cryptosystems **randomly pad the plaintext prior to encryption**. This **randomizes the ciphertext**.
- However, **more sophisticated CCAs are possible**, and a **simple padding** with a random value has been shown to be **insufficient to provide the desired security**.
- To **counter such attacks**, RSA Security Inc., a leading RSA vendor and former holder of the RSA patent, reco**mmends modifying the plaintext using a procedure known as optimal asymmetric encryption padding (OAEP).**

Thank You

Man-in-the-Middle (MitM) attack on the Diffie-Hellman Key Exchange