# Introdução à Chave Pública

- PK - Public Key
- RSA
- Exponenciação rápida
- Segurança RSA

# Compartilhamento de chaves

- Algoritmo de chave simétrica
  - Necessário compartilhar uma chave secreta previamente
  - Chave que encripta é a mesma que decripta
  - Problema: como compartilhar a chave?
- Algoritmo de chaves assimétricas
  - Temos 2 chaves: chave pública (encripta) e chave privada (decripta)
  - Fácil compartilhar a chave que encripta
  - Chave que encripta pode ser vista por qualquer um
  - Necessário guardar em segredo somente a chave privada

# RSA

- Criação por Rivest, Shamir, Adleman (1977)
- Define: algoritmo de encriptação $Enc_{pubK}(m)$ e de decriptação $Dec_{secK}(c)$
- É o sistema de chave pública mais popular
- Existem 3 tipos de sistemas assimétricos principais
- Patenteado nos EUA, expirado em 2000.

# Algoritmo RSA

- Geração de chave: necessário computar o par $(K_{pub}, K_{pr})$
- Algoritmo:
  1. Escolher primos $p$, $q$ grandes
  2. $n = p \times q$
  3. $\phi(n) = (p-1)(q-1)$ (função de Euler)
  4. Escolher $K_{pub} = e \in \{2, \ldots, \phi(n) - 1\}$ tal que

$$mdc(e, \phi(n)) = 1$$

  5. Computar $K_{pr} = d$ tal que

$$d \times e = 1 \mod \phi(n)$$

  6. Resultado: $K_{pub} = (n, e)$ e $K_{pr} = (d)$

# Função de Euler $\phi$

- $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$
- $\phi(n)$ =Quantos números em $\mathbb{Z}_n$ são coprimos com $n$?
  - $x$ é coprimo com $n$ se $mdc(n, x) = 1$
- Exemplo
  - $n = 6$, $\mathbb{Z}_n = \{0, 1, 2, 3, 4, 5\}$
  - $mdc(0, 6) = 6$, $mdc(1, 6) = 1$, $mdc(2, 6) = 2$, $mdc(3, 6) = 3$, $mdc(4, 6) = 2$, $mdc(5, 6) = 1$
  - $\phi(6) = 2$
- Como computar $\phi(n)$ eficientemente?
  - Se $n = p_1^{e_1} \times p_2^{e_2} \times \cdot p_k^{e_k}$ então

$$\phi(n) = \Pi_{i=1}^{k}(p_i^{e_i} - p_{i-1}^{e_i - 1})$$

  - Se $n = p \times q$ com $p, q$ primos, $\phi(n) = (p-1)(q-1)$
- Se os fatores primos de $n$ não forem conhecidos, não é possível computar $\phi(n)$ eficientemente

# O problema RSA

- Se $p$, $q$ são conhecidos:
  - então $\Phi(n)$ pode ser computado
  - então $d = e^{-1} \mod \Phi(n)$ pode ser computado
  - então possível computar $e$-ésima raízes módulo $n$
- Se $p$, $q$ não são conhecidos
  - então, computar $\Phi(n)$ é tão difícil quanto computar $n$
  - então computar $d$ é tão difícil quanto fatorar $n$

# Algoritmo RSA: observações

- $p, q \geq 2^{512}$
- $n \geq 2^{1024}$
- Nível de segurança está relacionado a $n$
  - Nem todos números são difíceis de fatorar
    - Metade dos números são par
    - 1/3 dos números são divisíveis por 3
  - Os números mais difíceis são produtos de 2 primos de tamanhos parecidos
  - É fácil testar se um número é primo (artigo: PRIMES in in P)

# RSA: encriptação e decriptação

- Encriptação
  - Dados $K_{pub} = (n, e)$, $x \in \mathbb{Z}_n = \{0, 1, \ldots, n-1\}$
    - $y = e_{K_{pub}}(x) = x^e \mod n$
- Decriptação
  - Dados $K_{pr} = d$, $y \in \mathbb{Z}_n$
    - $x = d_{K_{pr}}(y) = y^d \mod n$

# Exemplo

- Preparação:
  - $p = 3$, $q = 11$, $n = 33$, $\phi(n) = (3-1) \times (11-1) = 20$
  - Escolher $e = 3$ (concidência que é igual a $p$)
  - $d = e^{-1} = 7 \mod 20$
  - Verificando: $d \times e = 7 \times 3 = 1 \mod 20$
- Transmite $K_{pub} = (n, e) = (33, 3)$
- Encriptação:
  - Quer transmitir $x = 4$
  - $y = 4^3 = 64 \mod 33 = 31$
- Transmite $y$ (texto cifrado)
- Decriptação:
  - $x = y^d = 31^7 = 4 \mod 33$
  - $31^7 = (-2)^7 \mod 33 = -128 \mod 33$
  - $-128 \mod 33 = -4 \times 33 + 4 \mod 33 = 4$

# RSA: prova de corretude

- Esquema RSA:
  - $Enc_{K_{pub}=(n,e)}(x) = y = x^e \mod n$
  - $Dec_{K_{priv}=(n,d)}(y) = x = y^d \mod n$
- Provar que $d_{K_{pr}}(e_{K_{pub}}) \equiv (x^e)^d \equiv x^{ed} \equiv x \mod n$
- Por definição temos: $d \cdot e \equiv 1 \mod \Phi(n)$
- Para remover o mod , usamos um inteiro $t$

$$d \cdot e = 1 + t \cdot \Phi(n)$$

- Ou seja, escrito de outra forma queremos provar:
  $d_{K_{pr}}(y) = x^{de} \equiv x \mod n \Rightarrow x^{1+t\Phi(n)} \equiv x \mod n$
- E, de maneira equivalente, $x \mod n = \left(x^{\Phi(n)}\right)^t \cdot x \mod n$

# RSA: prova de corretude: continuação 1

- Queremos provar: $x \mod n = \left(x^{\Phi(n)}\right)^t \cdot x \mod n$
- Usamos o teorema de Euler:

$$mdc(a, b) = 1 \Rightarrow a^{\Phi(b)} \equiv 1 \mod b$$

  - Equivalente a: $\left(a^{\Phi(b)}\right)^t \equiv 1^t \mod b \equiv 1 \mod b$ ($t$ é int)
- Dois casos:
  - **Caso 1** (mais fácil): se $mdc(x, n) = 1$, aplicar Teo. Euler:

$$\left(x^{\Phi(n)}\right)^t \equiv 1 \mod n$$

e multiplicar por $x$ em ambos lados da equivalência:

$$x \cdot \left(x^{\Phi(n)}\right)^t \equiv x \cdot 1 \mod n \ \square$$

**provado caso 1!**

  - Caso 2 ... continua...

# RSA: prova de corretude: continuação 2

- Queremos provar: $x \mod n = \left(x^{\Phi(n)}\right)^t \cdot x \mod n$
- **Caso 2**: quando $mdc(x, n) = mdc(x, p \cdot q) \neq 1$.
  - Como $x < n$ e $p, q$ são primos, então para inteiros $r, s < p, q$ temos
    $x = r \cdot p$ ou $x = s \cdot q$
  - Se usarmos que $x = r \cdot p$, então $mdc(x, q) = 1$
    - Quando $x = s \cdot q$, a prova é similar
  - Então, com o teo. Euler:

$$x^{\Phi(q)} \equiv 1 \mod q \Rightarrow \left(x^{\Phi(q)}\right)^t \equiv 1 \mod q$$

  - Voltando ao termo $\left(x^{\Phi(n)}\right)^t$ temos, manipulando algebricamente essa equação, que:

$$\left(x^{\Phi(n)}\right)^t \equiv \left(x^{(q-1)(p-1)}\right)^t \equiv \left(\left(x^{\Phi(q)}\right)^t\right)^{p-1} \equiv 1^{(p-1)} \equiv 1 \mod q$$

  - Usaremos então que

$$\left(x^{\Phi(n)}\right)^t \equiv 1 \mod q$$

# RSA: prova de corretude: continuação 3 (caso 2)

- Queremos provar: $x \mod n = \left(x^{\Phi(n)}\right)^t \cdot x \mod n$
- Usaremos então que

$$\left(x^{\Phi(n)}\right)^t \equiv 1 \mod q$$

- Da definição de mod temos para um inteiro $u$:

$$\left(x^{\Phi(n)}\right)^t = 1 + uq$$

- Arrumando algebricamente:
  - $x\left(x^{\Phi(n)}\right)^t = (1 + uq)\, x$
  - $x\left(x^{\Phi(n)}\right)^t = x + \mathbf{x}uq$
  - $x\left(x^{\Phi(n)}\right)^t = x + \mathbf{rp}uq$, usando $\mathbf{x} = \mathbf{r} \cdot \mathbf{p}$
  - $x\left(x^{\Phi(n)}\right)^t = x + run$, usando $n = p \cdot q$
- Voltando ao mod $n$
  - $x\left(x^{\Phi(n)}\right)^t \equiv x + run \mod n$
  - $x\left(x^{\Phi(n)}\right)^t \equiv x \mod n$ □

**provado caso 2!**

# Exponenciação rápida

- Problema na prática é fazer
  - $y = x^e \mod n$ (encriptação)
  - $x = y^d \mod n$ (decriptação)

  com número extensos

# Exemplo

- $x^4 = ?$
- Uma forma
  - $x \times x = x^2$
  - $x^2 \times x = x^3$
  - $x^3 \times x = x^4$
  - Usamos 3 multiplicações
- Outra forma
  - $x \times x = x^2$
  - $x^2 \times x^2 = x^4$
  - Usamos 2 multiplicações
- E para $x^8 = ?$
- E para $x^{2^{1024}} = ?$

# Algoritmo "quadrado e multiplicar"

- "método binário" ou "exponenciação esquerda para direita"
- Exemplo:
  - $x^{26} = ?$
    $x \times x = x^2$ (quadrado)
    $x \times x^2 = x^3$ (multiplicar)
    $x^3 \times x^3 = x^6$ (quadrado)
    $x^6 \times x^6 = x^{12}$ (quadrado)
    $x \times x^{12} = x^{13}$ (multiplicar)
    $x^{13} \times x^{13} = x^{26}$ (quadrado)
- Usar a representação binária do expoente para escolher a operação certa
  - $x^{26} = x^{11010b}$
    $x^1 \times x^1 = x^{10b}$
    $x^1 \times x^{10b} = x^{11b}$
    $(x^{11b})^2 = x^{110b}$
    $(x^{110b})^2 = x^{1100b}$
    $x \times x^{1100b} = x^{1101b}$
    $(x^{1101b})^2 = x^{11010b}$

# Algoritmo “quadrado e multiplicar”

- Uma iteração por bit do exponente
- A cada iteração usar operação “quadrado”
- Se bit atual do exponente for 1b então “multiplicar”

# Gerando números primos

- Seja $\tilde{p}$ um número aleatório ímpar de $n$ bits
- A chance de $\tilde{p}$ ser primo é

$$P(\tilde{p} \text{ é primo}) \approx \frac{2}{\ln(\tilde{p})}$$

- Exemplo: $\tilde{p} \approx 2^{512}$
  - $\tilde{p}$ é primo com prob. $\frac{2}{\ln(2^{512})} = \frac{2}{512\ln(2)} \approx \frac{1}{177}$
- Para cada 177 números aleatórios ímpares de 512 bits, esperamos achar 1 primo

# Como verificar se $\tilde{p}$ é primo?

- Para testar de $\tilde{p}$ não é necessário fatorar
- Algoritmos eficientes atuais afirmam:
  - com certeza que números não são primos e
  - com alta probabilidade que são primos
- Parâmetro de segurança do teste determina a probabilidade de ser primo
- Teste de primalidade de Fermat
- Teste de Miller-Rabin

# Teste de primalidade de Fermat

**Algorithm 1** Algoritmo de teste de primalidade de Fermat

**Require:** candidato a primo $\tilde{p}$, parâmetro de segurança $s$

**Ensure:** afirma se $\tilde{p}$ é composto ou provavelmente primo

1: **for** $i \leftarrow 1 \ldots s$ **do**
2: Sortear $a \in \{2, 3, \ldots \tilde{p} - 2\}$
3: **if** $a^{\tilde{p}-1} \not\equiv 1 \mod \tilde{p}$ **then** ▷ Pequeno teorema de Fermat
4: **return** $\tilde{p}$ é composto
5: **end if**
6: **end for**
7: **return** provavelmente $\tilde{p}$ é primo

# Pequeno teorema de Fermat

- Se $p$ é primo e $a$ é um inteiro, então

$$a^{p-1} \equiv 1 \mod p$$

- Também escrito como

$$a^{p} \equiv a \mod p$$

- Porém
  - para alguns $p$ compostos (números de Carmichael), isso também é verdade

# Teste de Miller-Rabin

- Considere a seguinte decomposição de $\tilde{p}$, um ímpar candidato a primo, usando um $r$ inteiro ímpar e um $u$ inteiro qualquer:

$$\tilde{p} - 1 = 2^u r$$

- Se for possível achar inteiro $a$ tal que

$$a^r \not\equiv 1 \mod \tilde{p}$$

e

$$a^{r2^j} \not\equiv \tilde{p} - 1 \mod \tilde{p}$$

para todo $j = \{0, 1, \ldots, u\}$, então $\tilde{p}$ é composto, senão, provavelmente é primo.

# Teste de Miller-Rabin

**Algorithm 2** Algoritmo de teste de primalidade de Miller-Rabin

**Require:** candidato a primo $\tilde{p} = r2^u + 1$, parâmetro de seg. $s$

**Ensure:** afirma se $\tilde{p}$ é composto ou provavelmente primo

1: **for** $i \leftarrow 1 \ldots s$ **do**
2: Sortear $a \in \{2, 3, \ldots \tilde{p} - 2\}$
3: $z \equiv a^r \mod \tilde{p}$
4: **if** $z \not\equiv 1$ e $z \not\equiv \tilde{p} - 1$ **then**
5: **for** $j \leftarrow 1 \ldots u - 1$ **do**
6: $z \equiv z^2 \mod \tilde{p}$
7: **if** $z \equiv 1$ **then** ▷ $(\tilde{p}-1)^2 = \tilde{p}^2 - 1 \equiv -1 \equiv \tilde{p} - 1 \mod \tilde{p}$
8: **return** $\tilde{p}$ é composto
9: **end if**
10: **end for**
11: **if** $z \not\equiv \tilde{p} - 1$ **then**
12: **return** $\tilde{p}$ é composto
13: **end if**
14: **end if**
15: **end for**
16: **return** provavelmente $\tilde{p}$ é primo