The RSA Cryptosystem

→ Slower than symmetric ciphers

→ In practice, RSA is used for: Encryption of small pieces of data like keys and digital signatures → often used w/ symmetric ciphers like AES
↳ One-way f^n is the Integer Factorization Problem. Multiplying 2 large primes is easy → but factoring resulting product is hard

* x, y, n are very long numbers (2048+ bits)

→ RSA encryption & decryption is done in $\mathbb{Z}_n$ using modular arithmetic

→ RSA Encryption: Given public key $(n, e) = k_{pub}$ and $x \in \mathbb{Z}_n$, $y = e_{k_{pub}}(x) \equiv x^e \bmod n$
(public/encryption exponent)

→ RSA Decryption: Given private key $d = k_{pr}$ and $y \in \mathbb{Z}_n$, $x = d_{k_{pr}}(y) \equiv y^d \bmod n$ → Decryption / Private Exponent

→ If Alice wants to send encrypted message to Bob, she needs $k_{pub} = (n, e)$ so that Bob decrypts w/ $k_{pr} = d$

→ Requirements for RSA:
1. Attacker has k_{pub} ⇒ Computationally infeasible to find $d = k_{pr}$ w/ n & e
2. $x \in \mathbb{Z}_n$ ⇒ cannot encrypt more than ℓ = bit length of n bits
3. Should be easy for $x^e \bmod n$ and $y^d \bmod n$
4. For a given n, many public-private key pairs. Otherwise BFA possible

→ Practical RSA:

$p =$ E0DFD2C2A288ACEBC705EFAB30E4447541A8C5A47A37185C5A9CB98389CE4DE19199AA3069B404FD98C801568CB9170EB712BF10B4955CE9C9DC8CE6855C6123$_h$

$q =$ EBE0FCF21866FD9A9F0D72F7994875A8D92E67AEE4B515136B2A778A8048B149828AEA30BD0BA34B977982A3D42168F594CA99F3981DDABFAB2369F229640115$_h$

$n =$ CF33188211FDF6052BDBB1A37235E0ABB5978A45C71FD381A91AD12FC76DA0544C47568AC83D855D47CA8D8A779579AB72E635D0B0AAAC22D28341E998E90F82122A2C06090F43A37E0203C2B72E401FD06890EC8EAD4F07E686E906F01B2468AE7B30CBD670255C1FEDE1A2762CF4392C0759499CC0ABECFF008728D9A11ADF$_h$

$e =$ 40B028E1E4CCF07537643101FF72444A0BE1D7682F1EDB553E3AB4F6DD8293CA1945DB12D796AE9244D60565C2EB692A89B8881D58D278562ED60066DD8211E67315CF89857167206120405B08B54D10D4EC4ED4253C75FA74098FE3F7FB751FF5121353C554391E114C85B56A9725E9BD5685D6C9C7EED8EE442366353DC39$_h$

$d =$ C21A93EE751A8D4FBFD77285D79D6768C58EBF283743D2889A395F266C78F4A28E86F545960C2CE01EB8AD5246905163B28D0B8BAABB959CC03F4EC499186168AE9ED6D88058898907E61C7CCCC584D65D801CFE32DFC983707F87F5AA6AE4B9E77B9CE630E2C0DF05841B5E4984D059A35D7270D500514891F7B77B804BED81$_h$

→ Steps in computing public & private keys:

RSA Key Generation
Output: public key: $k_{pub} = (n, e)$ and private key: $k_{pr} = (d)$
1. Choose two large primes p and q.
2. Compute $n = p \cdot q$.
3. Compute $\Phi(n) = (p-1)(q-1)$.
4. Select the public exponent $e \in \{1, 2, \ldots, \Phi(n)-1\}$ such that

$$\gcd(e, \Phi(n)) = 1$$

5. Compute the private key d such that

$$d \cdot e \equiv 1 \bmod \Phi(n)$$

→ ensures e^{-1} exists modulo $\phi(n)$
∴ there is always private d

Ex: Alice wants to send msg to Bob. $x = 4$ ⟶ $k_{pub}(33, 3)$ → $y = 4^3 \bmod 33 \equiv 31 \bmod 33$

Bob: $p = 3$, $q = 11$, $n = p \cdot q = 33$
$\phi(n) = 2 \cdot 10 = 20$
choose $e = 3$ → $\gcd(e, \phi(n)) = 1$
$d \equiv e^{-1} \equiv 7 \bmod 20$

$k_{pr} = 7$ ⟶ $x \equiv 31^7 \bmod 33$
$x = 4$

→ Single Equation: $d_{k_{pr}}(y) = d_{k_{pr}}(e_{k_{pub}}(x)) \equiv (x^e)^d \equiv x^{ed} \equiv x \bmod n$

Choose $0 < e < \phi(n)$ s.t. $\gcd(e, \phi(n)) = 1$ → e is valid key

Chosen n ⇒ Use EEA w/ n and e: $\gcd(\phi(n), e) = s \cdot \phi(n) + t \cdot e$ (not required; inverse of e)

∴ $d \equiv t \bmod \phi(n)$

* if $\gcd(e, \phi(n)) \neq 1$, choose new e and repeat
* $e \cdot d \equiv 1 \bmod \phi(n)$

→ Correctness Proof: WTS that decryption is inverse of encryption: $d_{k_{pr}}(e_{k_{pub}}(x)) = x$ → Begin w/ construction rule for e and d:
$d \cdot e \equiv 1 \bmod \phi(n)$

Then $d \cdot e = 1 + t \cdot \phi(n)$ for $t \in \mathbb{Z}$

$d_{k_{pr}}(y) = x^{de} \equiv x^{1 + t \cdot \phi(n)} \equiv x \cdot x^{t \cdot \phi(n)} \equiv x \cdot (x^{\phi(n)})^t \bmod n$

WTS $x \equiv x \cdot (x^{\phi(n)})^t \bmod n$ ⟶ Use Eulers Theorem: if $\gcd(x, n) = 1$ then $1 \equiv x^{\phi(n)} \bmod n$
⇒ $1 \equiv 1^t \equiv (x^{\phi(n)})^t \bmod n$ for $t \in \mathbb{Z}$

Proof By Cases:

Case 1: $\gcd(x, n) = 1$. Euler's Theorem holds ⇒ $d_{k_{pr}}(y) \equiv (x^{\phi(n)})^t \cdot x \equiv 1 \cdot x \equiv x \bmod n$
* Establishes Decryption is inverse of Encryption for x relatively prime to $n = p \cdot q$

Case 2: $\gcd(x, n) = \gcd(x, p \cdot q) \neq 1$
We know p & q are primes ⇒ x must have one of them as a factor. → $x = r \cdot p$ or $x = s \cdot q$ for $r, s \in \mathbb{Z}$ s.t. $r < q$ and $s < p$

WLOG, assume $x = r \cdot p$ ⇒ $\gcd(x, q) = 1$. Euler's Theorem holds: $1 \equiv 1^t \equiv (x^{\phi(q)})^t \bmod q$ for any $t \in \mathbb{Z}^+$

$(x^{\phi(n)})^t \equiv (x^{(q-1)(p-1)})^t \equiv ((x^{\phi(q)})^t)^{p-1} \equiv 1^{p-1} = 1 \bmod q$

⇒ This is equivalent to $(x^{\phi(n)})^t = 1 + u \cdot q$ for some $u \in \mathbb{Z}$

$d_{k_{pr}}(y) \equiv x \cdot (x^{\phi(n)})^t \equiv (x + r \cdot u \cdot n) \bmod n \equiv x \bmod n$

∴ q.e.d.

→ Public Key Algs based on arithmetic w/ long numbers. → need more efficient computation methods.

$y = e_{k_{pub}}(x) \equiv x^e \bmod n$
$x = d_{k_{pr}}(y) \equiv y^d \bmod n$

Straight forward method:
$x \xrightarrow{SQ} x^2 \xrightarrow{MUL} x^3 \xrightarrow{MUL} x^4 \rightarrow \ldots$

→ Since e and d are very large, this is very slow ∴

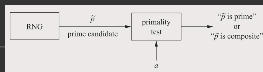
Solution: Square-And-Multiply Alg.
provides systematic way to find sequence of SQ and multiplications of $x \to x^H$

→ Intuition of SAM: Scanning bits from left → right. For every exponent bit, result is squared. Multiplication of result by x following SQ ≡ Currently scanned bit is 1
↳ After every operation, modular reduction performed.

Square-and-Multiply Algorithm for Modular Exponentiation
Input:
base element x
exponent $H = \sum_{i=0}^{t} h_i 2^i$, with $h_i \in 0, 1$ and $h_t = 1$
modulus n
Output: $x^H \bmod n$
Initialization: $r = x$
Algorithm:

1	FOR $i = t-1$ DOWNTO 0
1.1	$r \equiv r^2 \bmod n$
	IF $h_i = 1$
1.2	$r \equiv r \cdot x \bmod n$
2	RETURN (r)

→ To find large primes (p and q): Generate integers at random using an RNG and test for primality → Should be non-predictable



↳ How many random integers do I test before a prime?
↳ How fast can we test for primality?

Ex: RSA w/ 2048-bit n, $p, q \approx 2^{1024}$

As values increase, primes become less dense

$P(\tilde{p} \text{ is prime}) \approx \frac{2}{\ln(\tilde{p})}$

$P(\tilde{p} \text{ is prime}) \approx \frac{2}{\ln(2^{1024})} = \frac{2}{1024 \ln 2} \approx \frac{1}{354}$ → Must test 354 integers before 1 prime. → if primality test is fast, this is feasible.

→ Primality Tests are computationally easier than Factorization: If $\tilde{p} \in \mathbb{Z}$ is fed to Alg. $\tilde{p}$ is composite or prime
↳ To deal w/ false positives, test $\tilde{p}$ often w/ diff. values of a ⇒ likelihood that $(\tilde{p}, a)$ is wrong is small $< 2^{-80}$

→ Fermat's Primality Test:

$a^{\tilde{p}-1} \equiv 1 \bmod p$ holds for all primes

Fermat Primality Test
Input: prime candidate $\tilde{p}$ and security parameter s
Output: statement "$\tilde{p}$ is composite" or "$\tilde{p}$ is likely prime"
Algorithm:

1	FOR $i = 1$ TO s
1.1	choose random $a \in \{2, 3, \ldots, \tilde{p}-2\}$
1.2	IF $a^{\tilde{p}-1} \not\equiv 1 \bmod \tilde{p}$
1.3	RETURN ("$\tilde{p}$ is composite")
2	RETURN ("$\tilde{p}$ is likely prime")

Carmichael Numbers are composites that act like primes
$a^{C-1} \equiv 1 \bmod C$ → very rare.

→ Sol^n: Miller-Rabin Test

→ Miller-Rabin Test based on following theorem
→ Theorem: Given decomposition of an odd prime candidate $\tilde{p} - 1 = 2^u r$ → odd
↳ If we can find $a \in \mathbb{Z}$ s.t. $a^r \not\equiv 1 \bmod \tilde{p}$ and $a^{r 2^j} \not\equiv \tilde{p} - 1 \bmod \tilde{p}$ for all $j \in \mathbb{Z}_u$, $\tilde{p}$ is composite. $\tilde{p}$ is prime o.w.

Miller–Rabin Primality Test
Input: prime candidate $\tilde{p}$ with $\tilde{p} - 1 = 2^u r$ and security parameter s
Output: statement "$\tilde{p}$ is composite" or "$\tilde{p}$ is likely prime"
Algorithm:

1	FOR $i = 1$ TO s
	choose random $a \in \{2, 3, \ldots, \tilde{p}-2\}$
1.2	$z \equiv a^r \bmod \tilde{p}$
1.3	IF $z \not\equiv 1$ AND $z \not\equiv \tilde{p}-1$
	$j = 1$
1.4	WHILE $j \leq u-1$ AND $z \not\equiv \tilde{p}-1$
	$z \equiv z^2 \bmod \tilde{p}$
1.5	IF $z \equiv 1$ RETURN ("$\tilde{p}$ is composite")
	ELSE $j = j+1$
1.6	IF $z \not\equiv \tilde{p}-1$ RETURN ("$\tilde{p}$ is composite")
2	RETURN ("$\tilde{p}$ is likely prime")

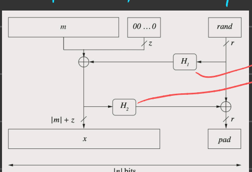
→ School Book RSA has problems.
↳ Encryption is deterministic → Specific plaintext always mapped to same ciphertext
↳ $x = 0, 1, -1$ ⇒ $y = 0, 1, -1$
↳ Small e and x subject to attacks if no padding used
↳ RSA is malleable.

Oscar can transform y into y' that leads to known transformation of x
* Oscar doesn't decrypt y but is capable of manipulating x

$y \to s^e y$ for $s \in \mathbb{Z}$ ⇒ $(s^e y)^d \equiv s^{ed} x^{ed} \equiv sx \bmod n$

→ Sol^n: Optimal Asymmetric Encryption Padding (OAEP)
embeds random structure into x before encryption.

→ Optimal Asymmetric Encryption Padding:



Mask Generation

H_1 and H_2 are hash f^n w/ lengths $|m| + z$ and r

→ 2 strings added to message m before encryption
↳ rand consisting of r bits → turns RSA from Deterministic → Probabilistic Encryption Scheme
↳ All-zero string consisting of z-bits to prevent malleability

$|n| = |m| + z + r$ ⟶ $|m| < |n|$

RSA Encryption with OAEP
Given a message m and the public key $k_{pub} = (n, e)$, the encryption function is:

$$y = e_{k_{pub}}(m) \equiv (x \| pad)^e \bmod n$$

(concat.)

where

$$x = (m \| 00\ldots0) \oplus H_1(rand)$$
$$pad = rand \oplus H_2(x)$$

RSA Decryption with OAEP
Given the private key $k_{pr} = d$ and the ciphertext y, the decryption process is:

1. RSA decryption: $d_{k_{pr}}(y) = y^d \equiv x \| pad \bmod n$
2. recompute $rand = H_2(x) \oplus pad$
3. recompute $m \| 00\ldots0 = x \oplus H_1(rand)$
4. verify that the zero string from Step 3 in fact contains z zero bits

Forms 2-round Feistel Network
First = rand value encrypts m and z 0 bits
Second takes output of R1 $(m \| 00\ldots0) \oplus H_1(rand)$ to encrypt rand itself

→ Possible Attacks:
1. Protocol Attack: Exploit RSA's weaknesses like malleability (solved w/ padding)
↳ Security Protocols must be implemented if RSA is used
2. Mathematical Attack: Factoring modulus. ⇒ Oscar knows n, e and y → wants d

He knows $e \cdot d \equiv 1 \bmod \phi(n)$ → can apply EEA to find d.
$\phi(n) = (p-1)(q-1)$
$d^{-1} \equiv e \bmod \phi(n)$
$x \equiv y^d \bmod n$

Solve issue by issuing larger modulus → Reason for why RSA must be at least 2048 bits.

3. Side-Channel Attacks