2550 Intro to cybersecurity

Public key Crypto

abhi shelat

Basic Number theory for PKE

Two main ideas:

Extended Greatest Common Divisor (Euclid)
Euler’s theorem

Modular Exponentiation

$(a, x, n) \rightarrow a^x \bmod n$

$7^{19} \bmod 31$

$$\underbrace{7 \cdot 7 \cdots 7}_{19 \text{ times}}$$

Modular Exponentiation

$(a, x, n) \rightarrow a^x \bmod n$

$7^{19} \bmod 31$

Modular Exponentiation

$(a, x, n) \rightarrow a^x \bmod n$

$7^{19} \bmod 31$

Modular Exponentiation $(a, x, n) \rightarrow a^x \bmod n$

$7^{19} \bmod 31$

$7^1 \quad 7^2 \quad 7^4 \quad 7^8 \quad 7^{16} \pmod{31}$

Modular Exponentiation

$(a, x, n) \rightarrow a^x \bmod n$

$$7^{19} \bmod 31$$

$$7^1 \quad 7^2 \quad 7^4 \quad 7^8 \quad 7^{16} \pmod{31}$$

7 18 14 10 7

Modular Exponentiation

$$(a, x, n) \rightarrow a^x \bmod n$$

Algorithm 2: ModularExponentiation(a, x, n)

Input: $a, x \in [1, n]$

$r \leftarrow 1$

while $x > 0$ **do**

  if x *is odd* **then**

    $r \leftarrow r \cdot a \bmod n$

  $x \leftarrow \lfloor x/2 \rfloor$

  $a \leftarrow a^2 \bmod n$

Return r

Modular Exponentiation

$$(a, x, n) \rightarrow a^x \bmod n$$

$$a^x \bmod n = \prod_{i=0}^{\ell} x_i a^{2^i} \bmod n$$

Algorithm 2: ModularExponentiation(a, x, n)

Input: $a, x \in [1, n]$

1 $r \leftarrow 1$
2 **while** $x > 0$ **do**
3 **if** x *is odd* **then**
4 $r \leftarrow r \cdot a \bmod n$
5 $x \leftarrow \lfloor x/2 \rfloor$
6 $a \leftarrow a^2 \bmod n$
7 Return r

Greatest Common Divisor

GCD(A,B) = GCD( )

Greatest Common Divisor

$$GCD(A,B) = GCD(B, A \bmod B)$$

Greatest Common Divisor

GCD(6809,1641)

Greatest Common Divisor

GCD(6809, 1641)	6809	=	4	·	1641	+ 245
GCD(1641, 245)	1641	=	6	·	245	+ 171
GCD(245, 171)	245	=	1	·	171	+ 74
GCD(171, 74)	171	=	2	·	74	+ 23
GCD(74, 23)	74	=	3	·	23	+ 5
GCD(23, 5)	23	=	4	·	5	+ 3
GCD(5, 3)	5	=	1	·	3	+ 2
GCD(3, 2)	3	=	1	·	2	+ **1**
GCD(2, 1)	2	=	2	·	1	+ 0
GCD(1, 0) **= 1**	remainder 0: stop					

Each step

$a = q \cdot b + r$

$GCD(a, b) = GCD(b, r)$

Stop

when r = 0. The last nonzero remainder is the GCD.

Answer

GCD(6809, 1641) = 1

4 Quotients, reused later for the extended GCD

Extended GCD: Back-Substitution

Start from 3 = 1·2 + 1 and work upward: each step replaces the smaller number using the row above it

$$1 = 3 - \mathbf{1}\cdot 2$$

$$= 3 - 1\cdot(5 - \mathbf{1}\cdot 3)$$

$$= 2\cdot 3 - 1\cdot 5$$

Extended GCD: Back-Substitution

Start from 3 = 1·2 + 1 and work upward: each step replaces the smaller number using the row above it

$$
\begin{aligned}
1 &= 3 - \mathbf{1}\cdot 2 \\
&= 3 - 1\cdot(5 - \mathbf{1}\cdot 3) \\
&= 2\cdot(23 - \mathbf{4}\cdot 5) - 1\cdot 5 \\
&= 2\cdot 23 - 9\cdot(74 - \mathbf{3}\cdot 23) \\
&= 29\cdot(171 - \mathbf{2}\cdot 74) - 9\cdot 74 \\
&= 29\cdot 171 - 67\cdot(245 - \mathbf{1}\cdot 171) \\
&= 96\cdot(1641 - \mathbf{6}\cdot 245) - 67\cdot 245 \\
&= 96\cdot 1641 - 643\cdot(6809 - \mathbf{4}\cdot 1641) \\
&= 2\cdot 3 - 1\cdot 5 \\
&= 2\cdot 23 - 9\cdot 5 \\
&= 29\cdot 23 - 9\cdot 74 \\
&= 29\cdot 171 - 67\cdot 74 \\
&= 96\cdot 171 - 67\cdot 245 \\
&= 96\cdot 1641 - 643\cdot 245 \\
&= \mathbf{2668\cdot 1641 - 643\cdot 6809}
\end{aligned}
$$

So $2668\cdot 1641 \equiv 1 \pmod{6809}$, i.e. $1641^{-1} \bmod 6809 = \mathbf{2668}$

Same computation gives RSA's private exponent $d = e^{-1} \bmod \varphi(N)$

given (a,b), finds (x,y) s.t.
ax + by = gcd(a,b)

Algorithm 1: ExtendedEuclid(a, b)

Input: (a, b) s.t $a > b \geq 0$
Output: (x, y) s.t. $ax + by = \gcd(a, b)$

```
if a mod b = 0 then
    Return (0, 1)
else
    (x, y) ← ExtendedEuclid (b, a mod b)
    Return (y, x − y(⌊a/b⌋))
```

Greatest Common Divisor

GCD(6809,1641)

GCD allows us to compute modular inverses.

6809*(-643) + 1641*2668 =

Greatest Common Divisor

GCD(6809,1641)

```
6809*(-643) + 1641*2668 =   1

  -437818       437818
```

GCD allows us to compute modular inverses.

Main point

If $\gcd(x, y) = 1$, then it is easy to compute values (a, b) such that:

$$a * x + b * y = 1$$

The value a is called the “modular inverse” of x mod y.

Euler totient function



$$\phi(n)$$

“number of positive integers up to n that are relatively prime to n”

Euler totient

$\phi(15) =$

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Euler totient

prime

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

product of 2 primes

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

Euler theorem

$$\forall a \in \mathbb{Z}_n^\star,\ a^{\Phi(n)} = 1 \bmod n$$

Examples

$$7^{30} \bmod 31 =$$

1	2	4	8	16
7	18	14	10	7

Examples

$$2^8 \bmod 15 =$$

Implications of Euler

$$a^{10\phi(N)} \bmod N =$$

$$a^{k\phi(N)+1} \bmod N =$$

compute

$$11^{30^{2021}} \bmod 23$$

(show your work)

“Textbook” RSA (insecure)

Pick N = p*q where p,q are primes.

“Textbook” RSA (insecure)

Pick N = p*q where p,q are primes.

Pick e,d such that $e \cdot d = 1 \mod \phi(N)$

“Textbook” RSA (insecure)

Pick N = p*q where p,q are primes.

Pick e,d such that $e \cdot d = 1 \mod \phi(N)$

$\text{Enc}_{N,e}(m) = m^e \mod N$

$\text{Dec}_{N,d}(c) = c^d \mod N$

“Textbook” RSA (insecure)

Pick N = p*q where p,q are primes.

Pick e,d such that $e \cdot d = 1 \mod \phi(N)$

$\text{Enc}_{N,e}(m) = m^e \mod N$

$\text{Dec}_{N,d}(c) = c^d \mod N$

$(m^e)^d \mod N =$

Example of Textbook RSA

m=5

PK = (N=143, e=7) SK = (d=103)

“Textbook” RSA (insecure)

Pick N = p*q where p,q are primes.

Pick e,d such that $e \cdot d = 1 \mod \phi(N)$

$\text{Enc}_{N,e}(m) = m^e \mod N$

$\text{Dec}_{N,d}(c) = c^d \mod N$

Why is it insecure against IND-CPA attack?

pkcs1.5

ENC$_{pk}(m)$

PICK r AS A RANDOM STRING WITH NO 0s

(TYPICALLY 8 BYTES)

$$c \leftarrow (0||2||r||0||m)^e \bmod N$$

"PADDING ORACLE" ATTACK AGAINST THIS SCHEME

RSA-OAEP+

$\textsc{Gen}(1^n)$

$f, f^{-1} \leftarrow \textsc{trapdoor owp}()$

$\textsc{Enc}_{pk}(m)$

$r \leftarrow U_n$

$s \leftarrow R_1(r) \oplus m \;||\; R_2(r||m)$

$t \leftarrow R_3(s) \oplus r$

$c \leftarrow f(s||t)$

$R_1 : \{0,1\}^{k_0} \rightarrow \{0,1\}^n$

$R_2 : \{0,1\}^{n+k_0} \rightarrow \{0,1\}^{k_1}$

$R_3 : \{0,1\}^{n+k_1} \rightarrow \{0,1\}^{k_0}$

$\textsc{Dec}_{sk}(c)$

$(s = (s_1, s_2), t) \leftarrow f^{-1}(c)$

$r \leftarrow R_3(s) \oplus t$

$m \leftarrow R_1(r) \oplus s_1$

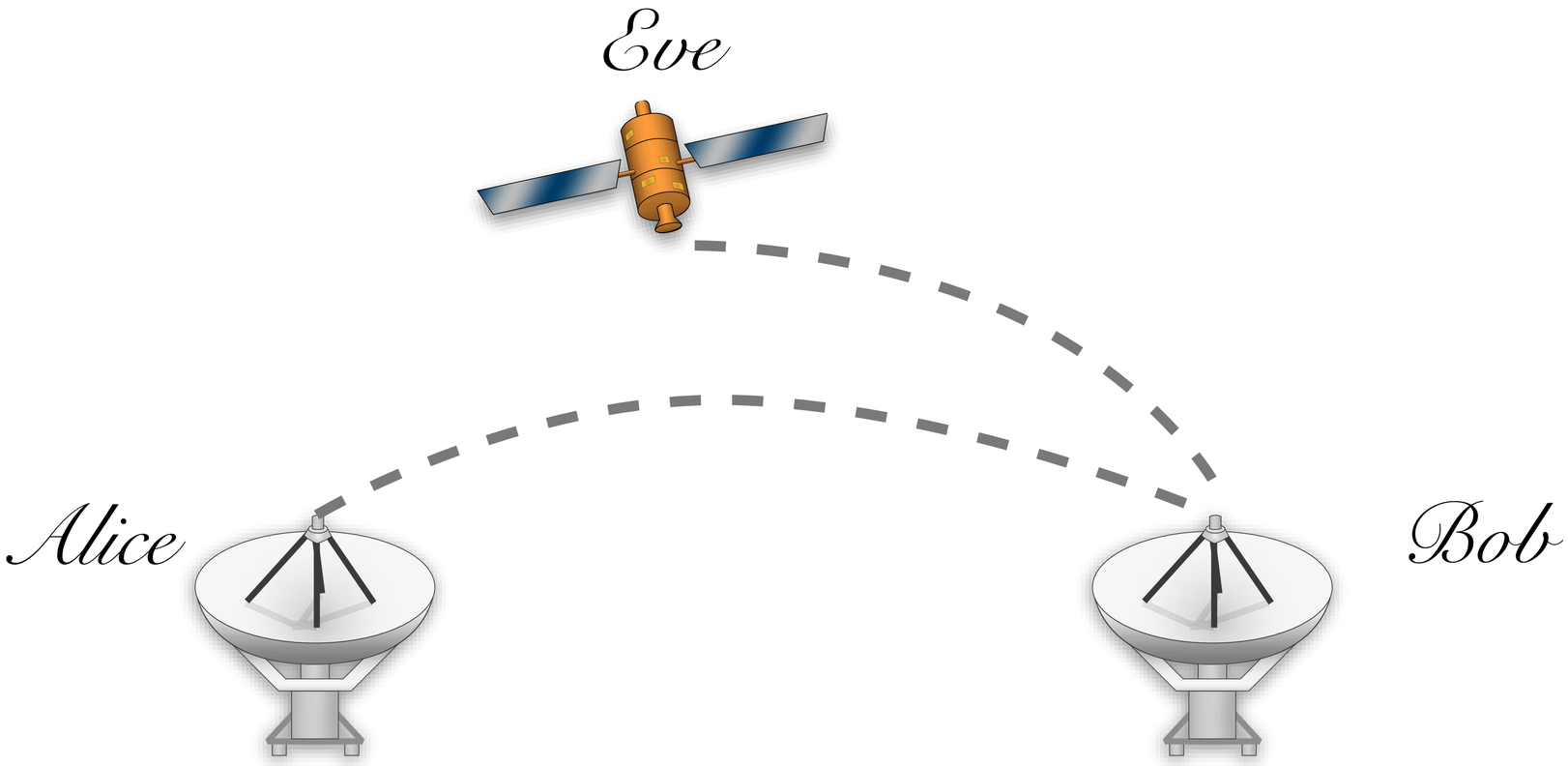
$R_2(r||m) \stackrel{?}{=} s_2$ $\quad$ $\textsc{output } m \textsc{ else fail}$

Very old problem

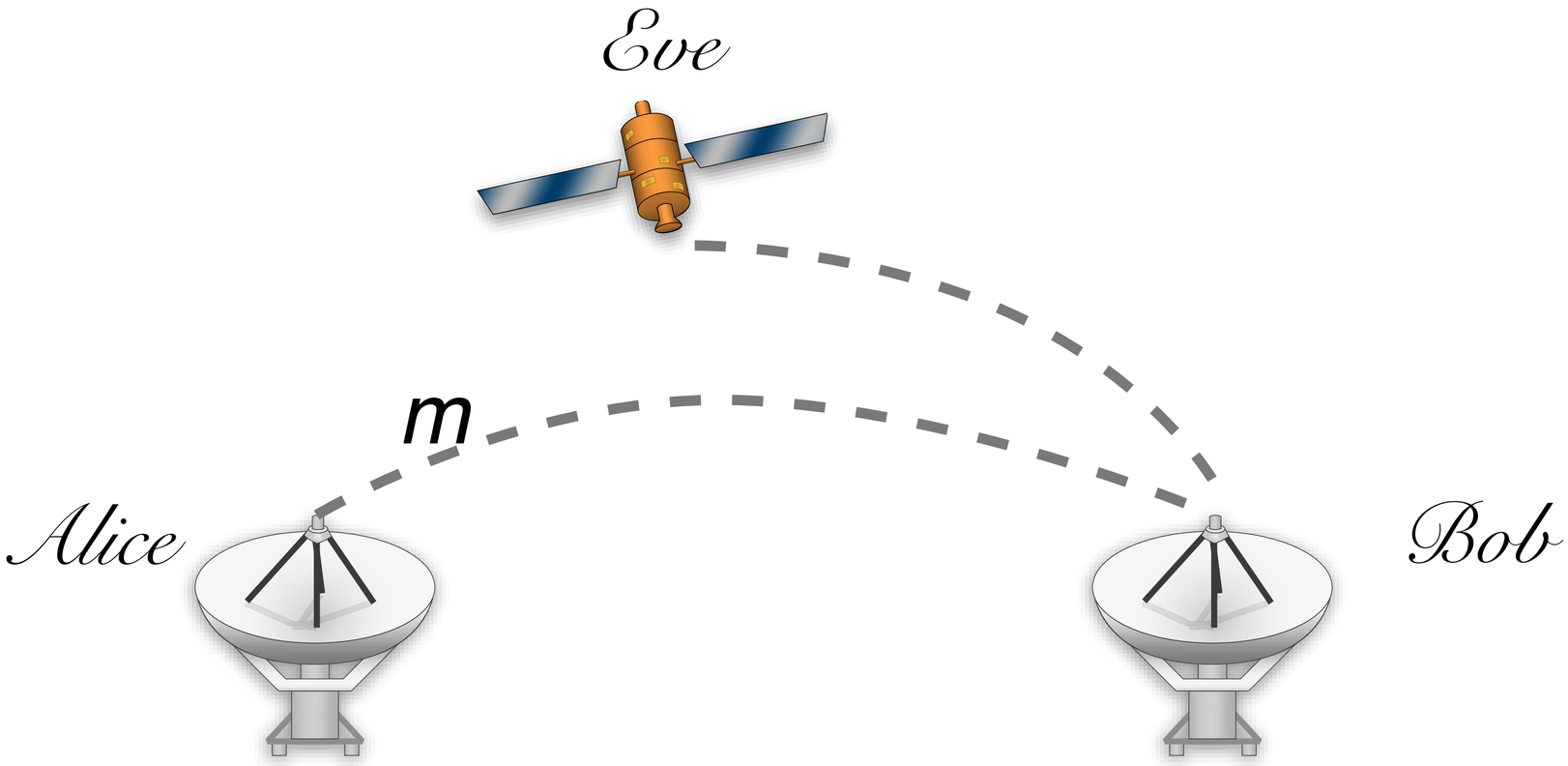




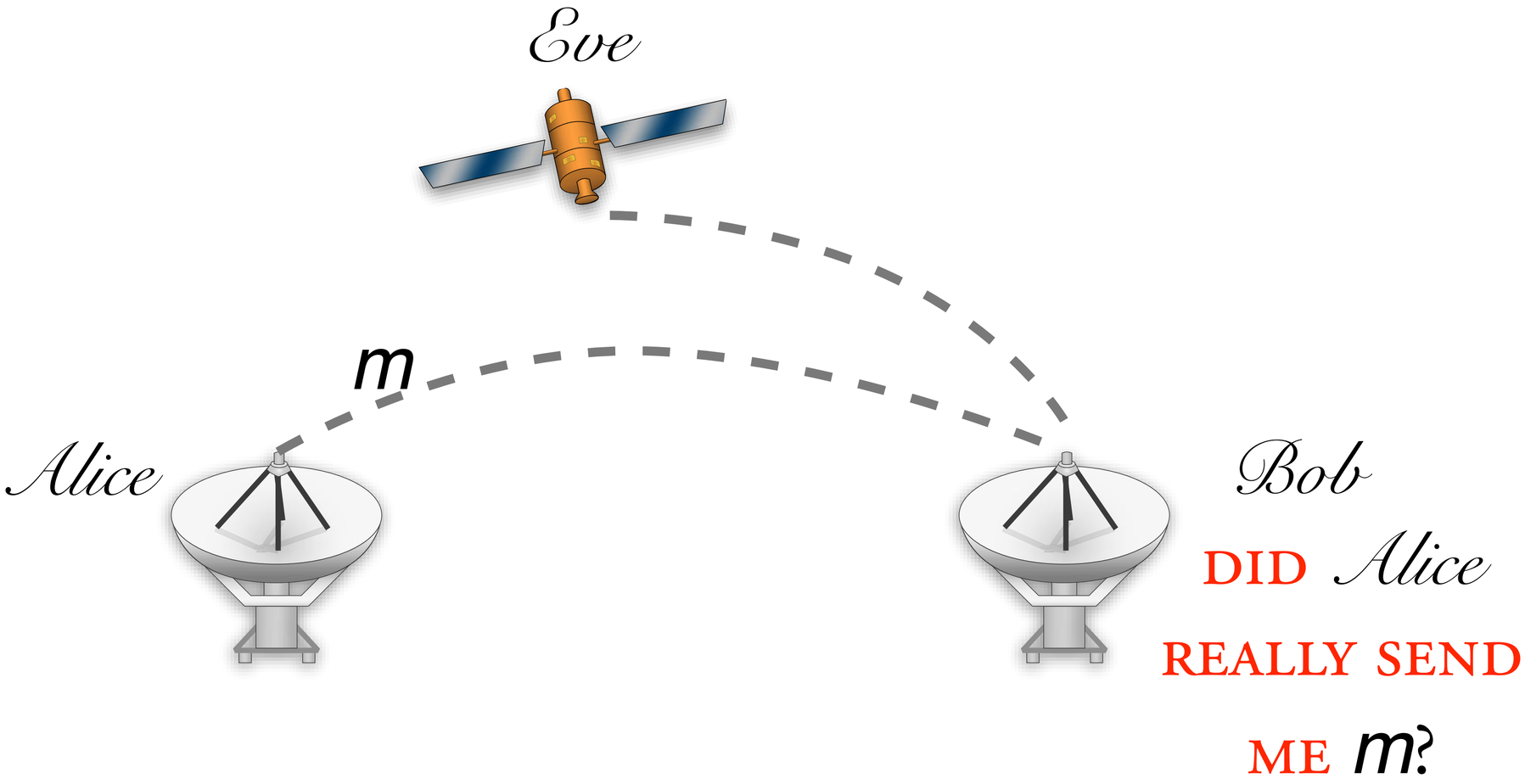
New Problem



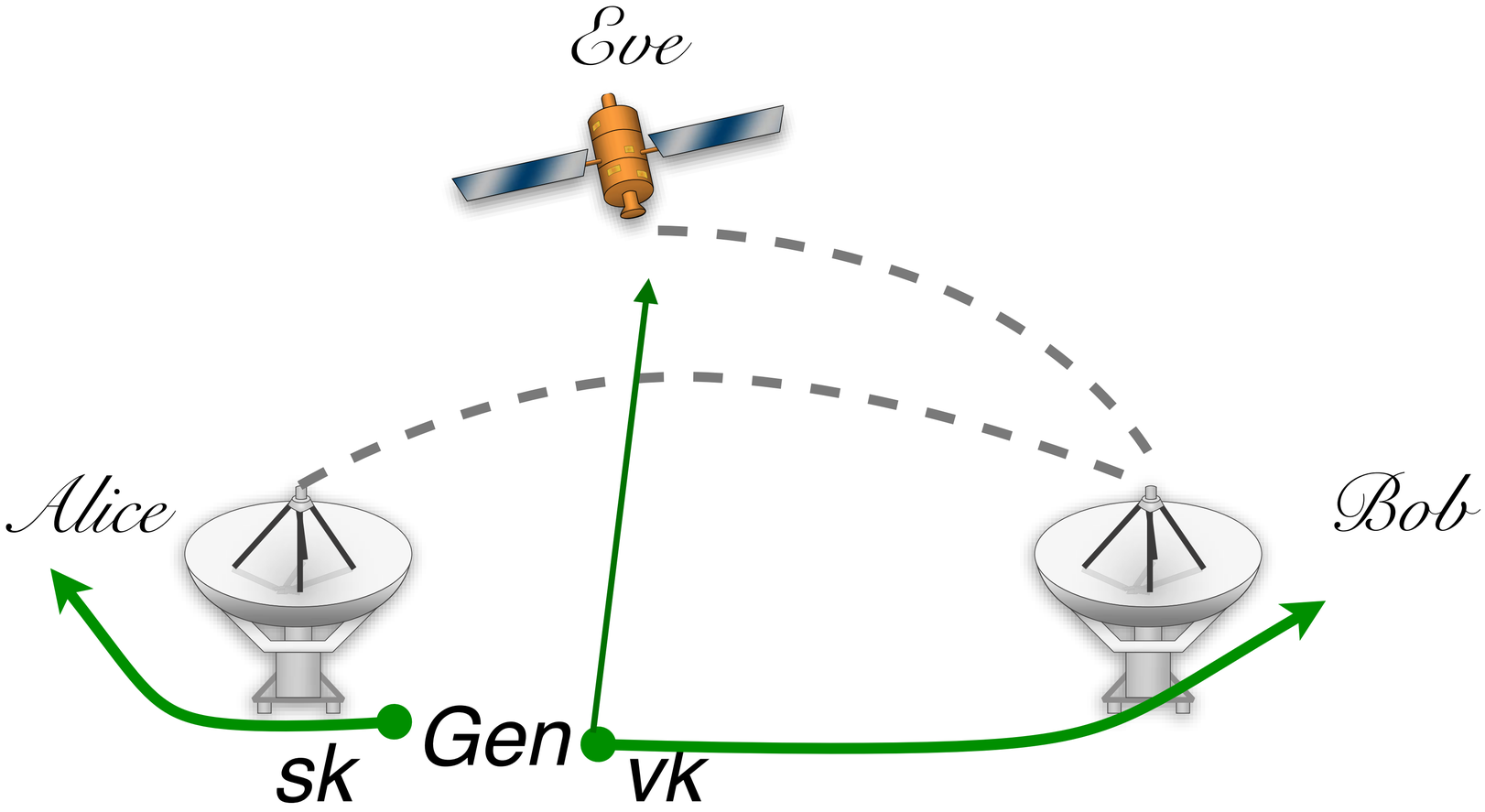
New Problem



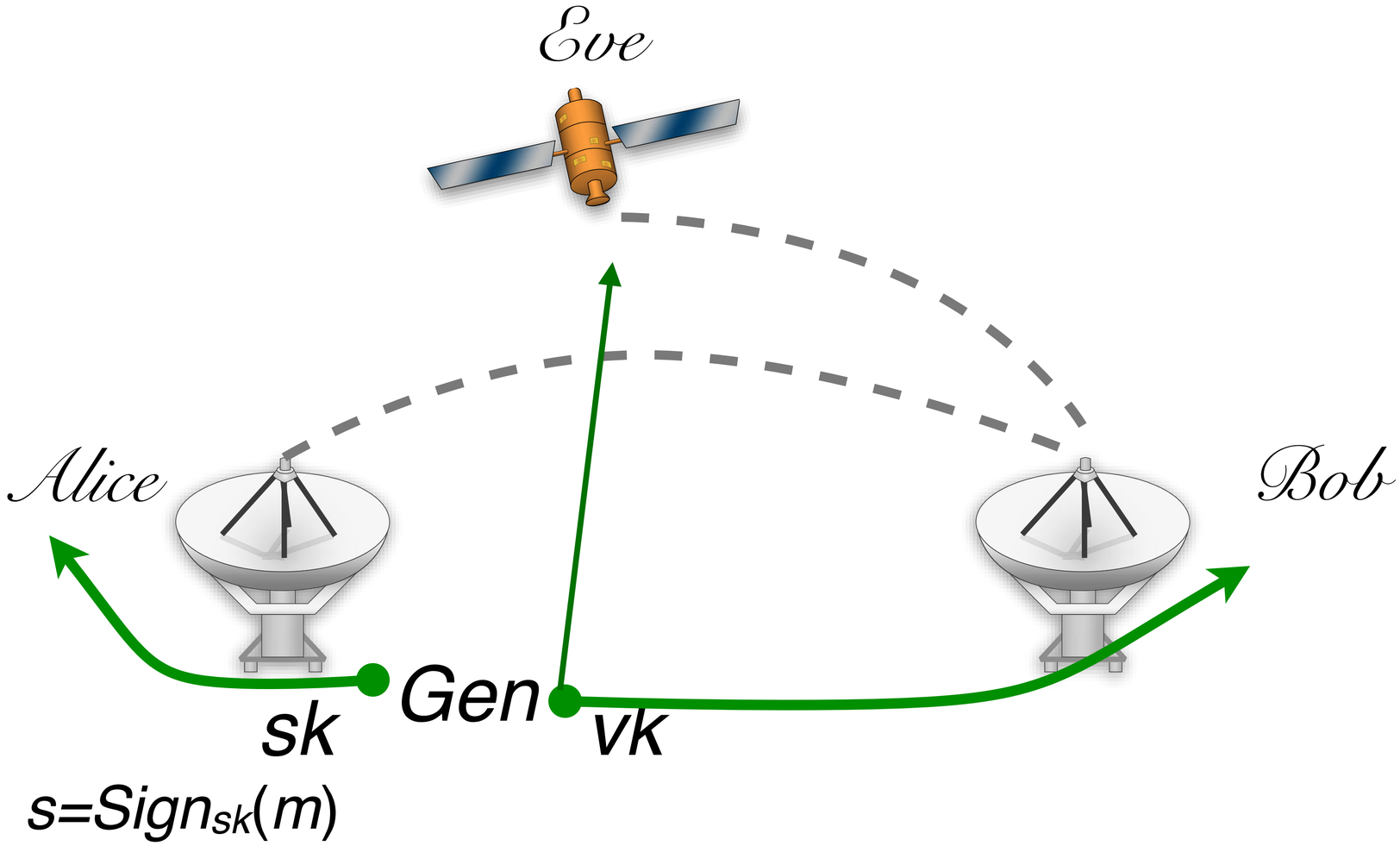
New Problem



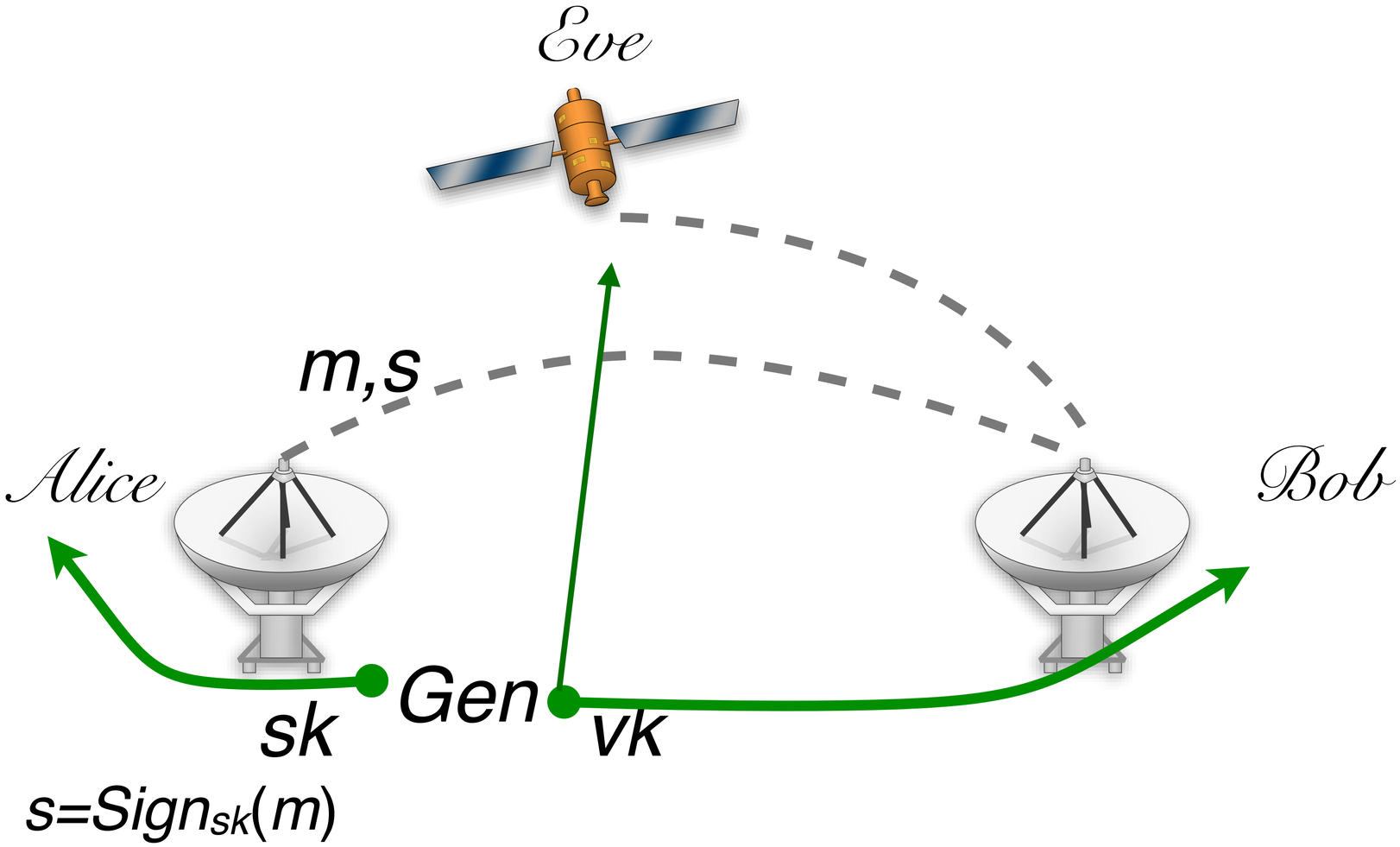
Public key digital signature



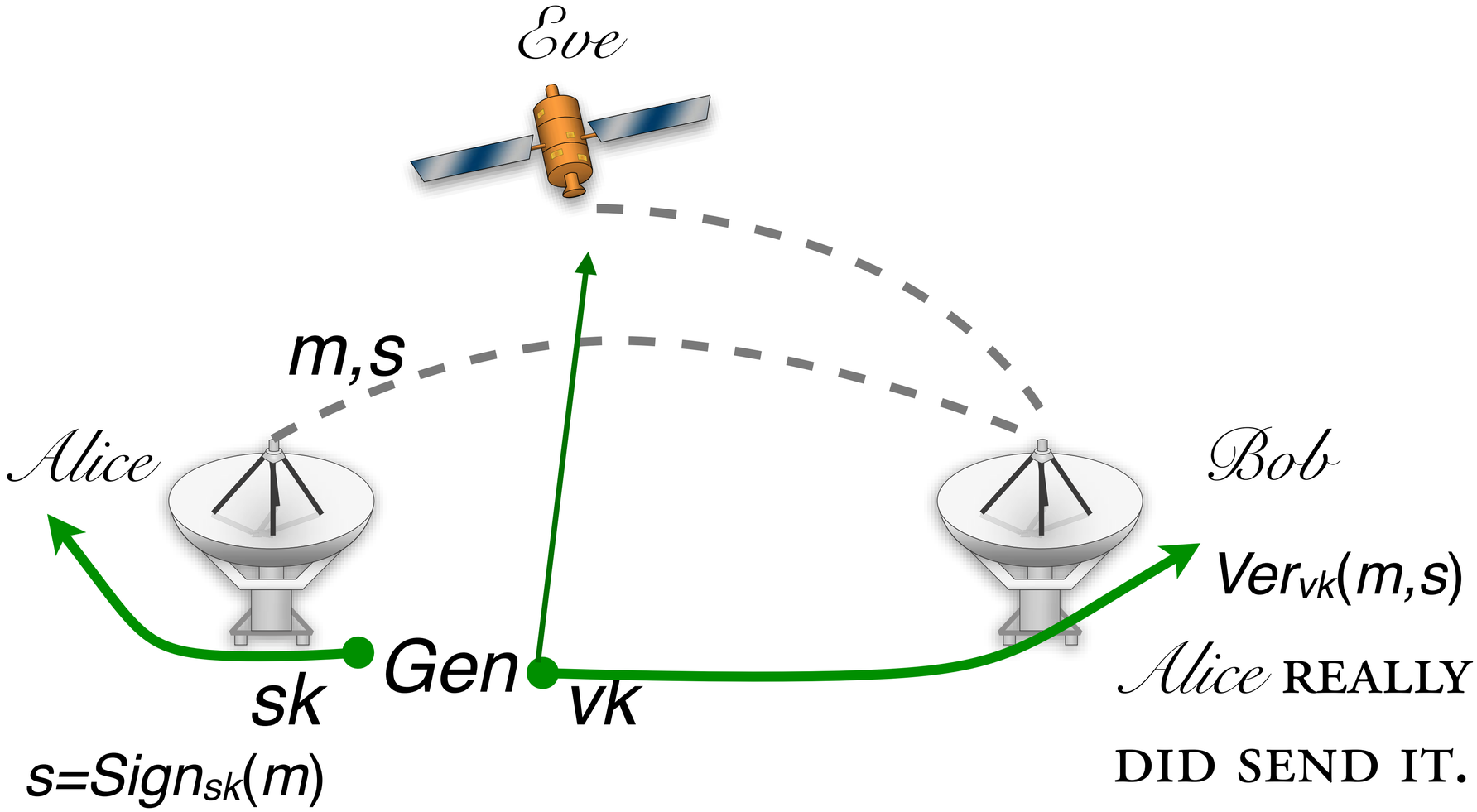
Public key digital signature



Public key digital signature



Public key digital signature



Public key digital signature

MESSAGE SPACE $\{\mathcal{M}\}_n$

$Gen(1^n)$

$Sign_{sk}(m)$

$Ver_{vk}(m,s)$

Public key digital signature

MESSAGE SPACE $\{\mathcal{M}\}_n$

$Gen(1^n)$ GENERATES A KEY PAIR sk, vk

$Sign_{sk}(m)$

$Ver_{vk}(m,s)$

Public key digital signature

MESSAGE SPACE $\{\mathcal{M}\}_n$

$Gen(1^n)$ GENERATES A KEY PAIR sk, vk

$Sign_{sk}(m)$ GENERATES A SIGNATURE s FOR $m \in \mathcal{M}_n$

$Ver_{vk}(m,s)$

Public key digital signature

MESSAGE SPACE $\{\mathcal{M}\}_n$

$Gen(1^n)$ GENERATES A KEY PAIR ***sk,vk***

$Sign_{sk}(m)$ GENERATES A SIGNATURE ***s*** FOR $m \in \mathcal{M}_n$

$Ver_{vk}(m,s)$ ACCEPTS OR REJECTS A MSG,SIG PAIR

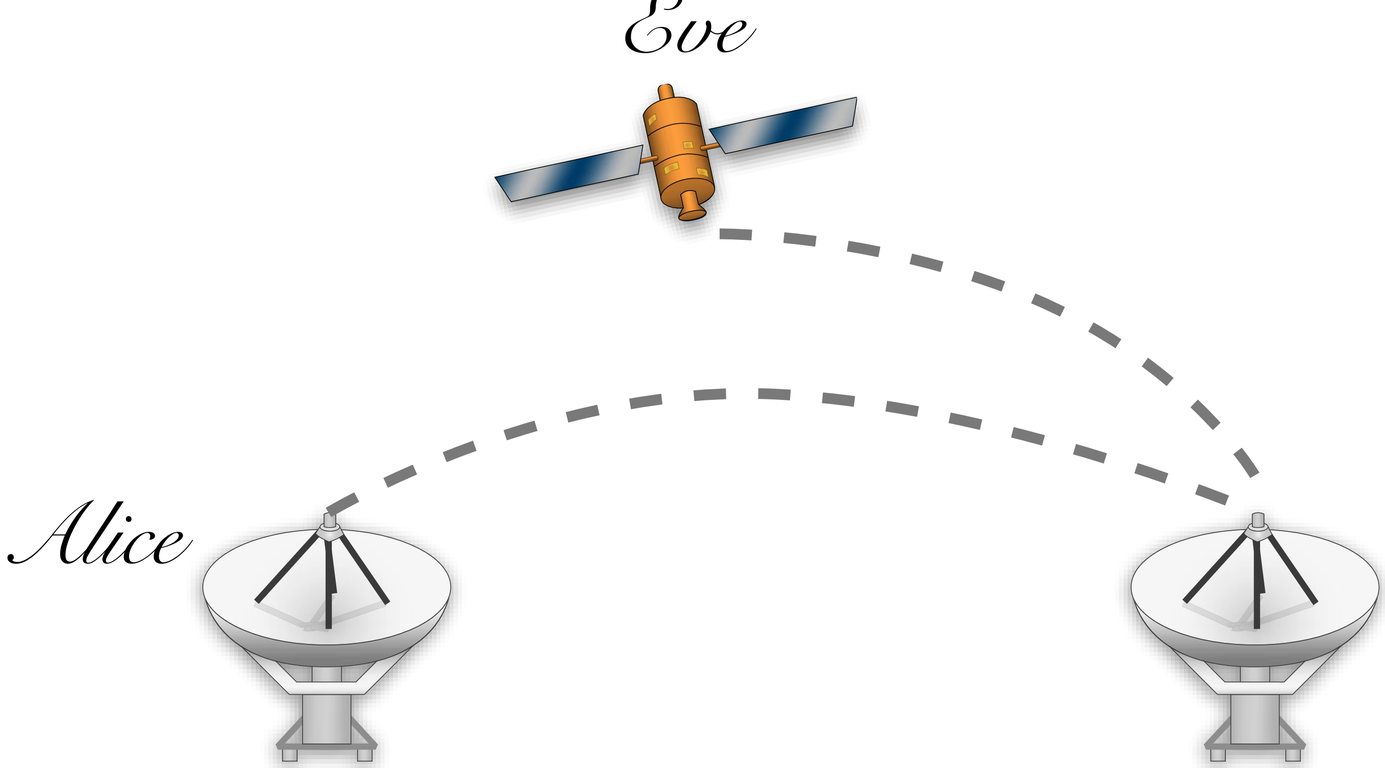
$$\Pr[k \leftarrow Gen(1^n) : Ver_{vk}(m, Sign_{sk}(m)) = 1] = 1$$

existential unforgability

"EVEN WHEN GIVEN A SIGNING ORACLE,

AN ADVERSARY CANNOT FORGE A SIGNATURE FOR

ANY MESSAGE OF ITS CHOOSING "

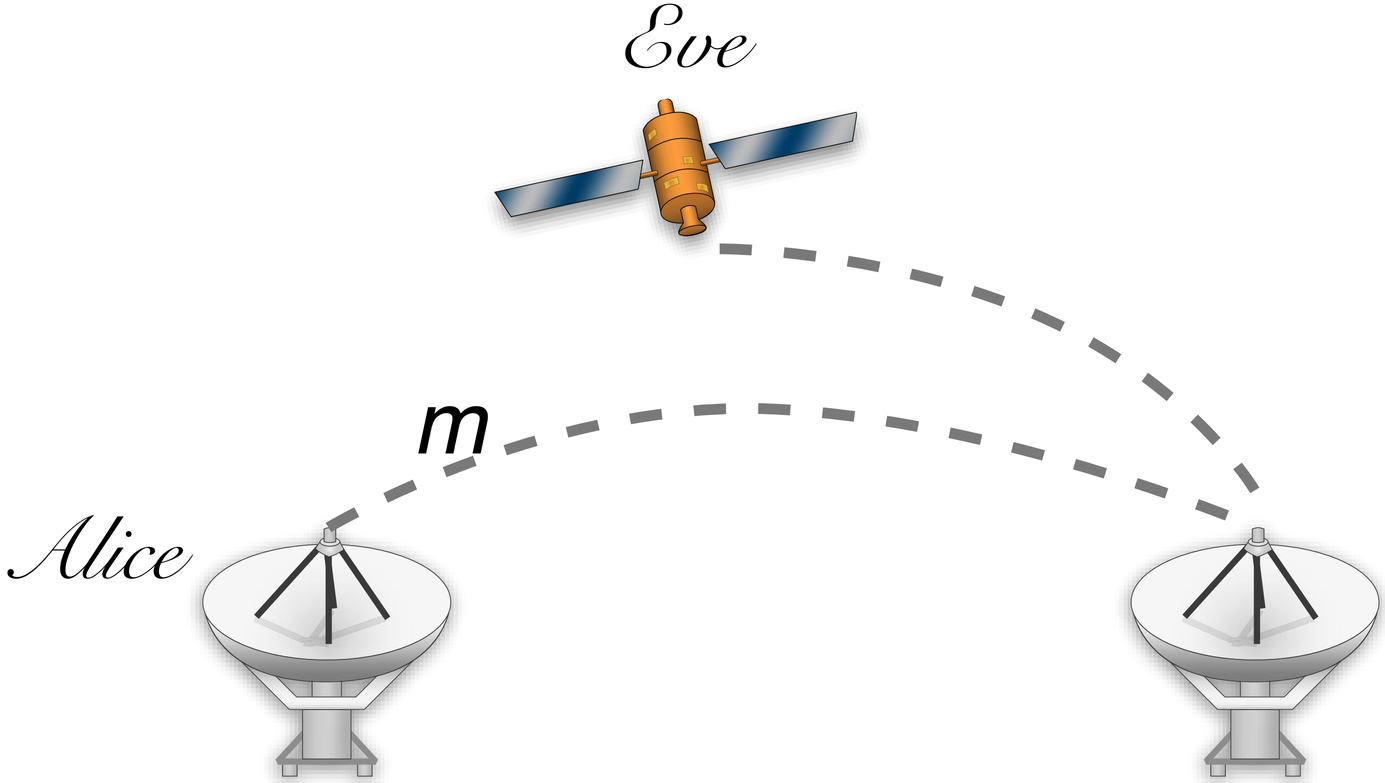


existential unforgability

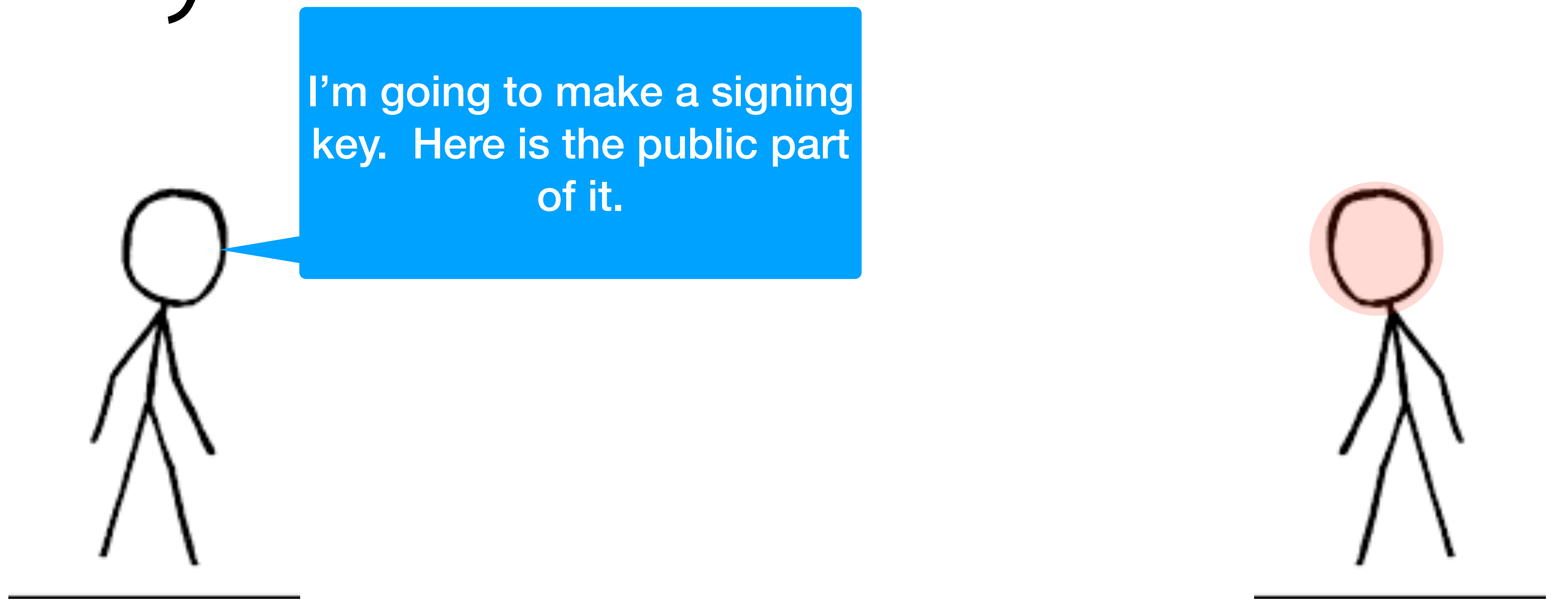
"EVEN WHEN GIVEN A SIGNING ORACLE,

AN ADVERSARY CANNOT FORGE A SIGNATURE FOR

ANY MESSAGE OF ITS CHOOSING "



Signature security



$(vk, sk) \leftarrow \mathrm{Gen}(1^n)$

Signature security



$(vk, sk) \leftarrow \mathrm{Gen}(1^n)$

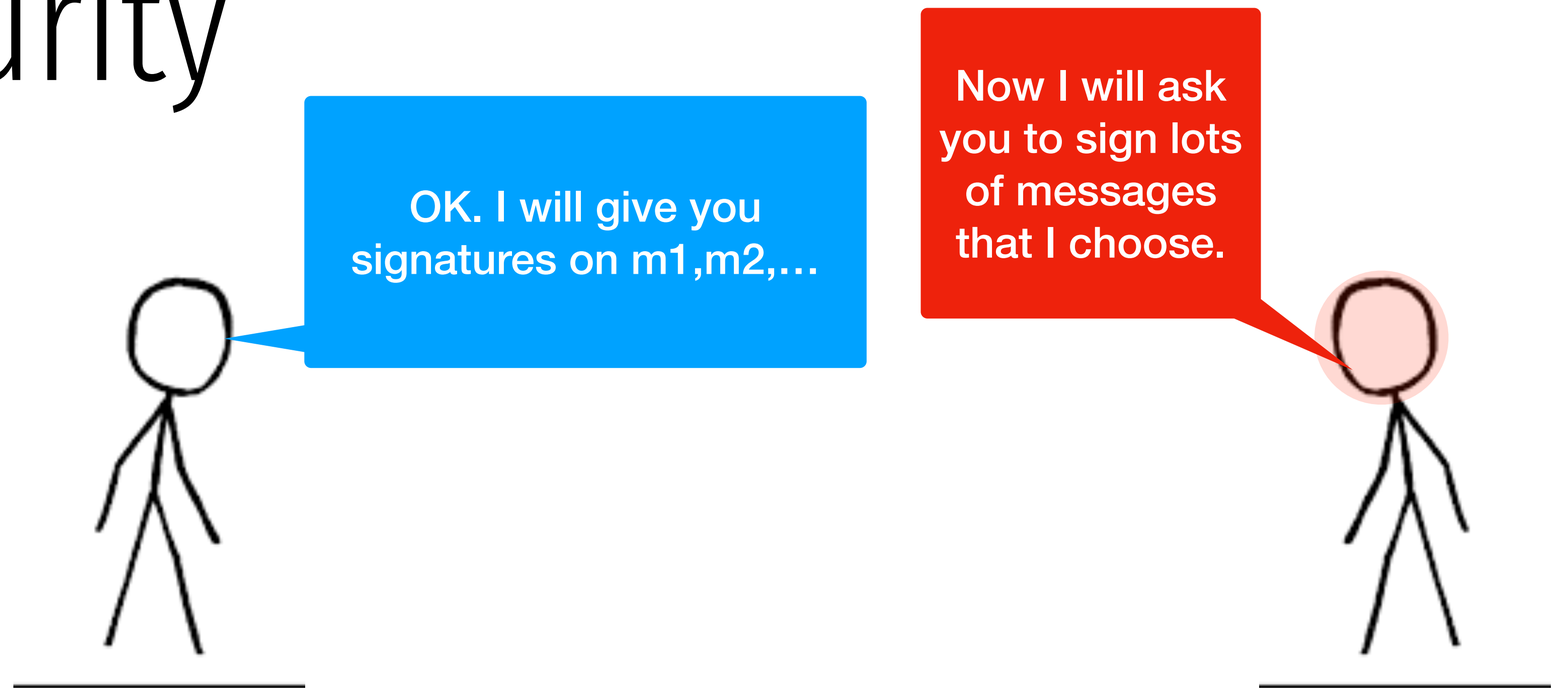
Now I will ask you to sign lots of messages that I choose.

$m_0, m_1, \ldots$



vk

Signature security



Signature security

Now I will try to create a new (signature, message) pair…one that I didn't receive from yoiu. signature on a new message





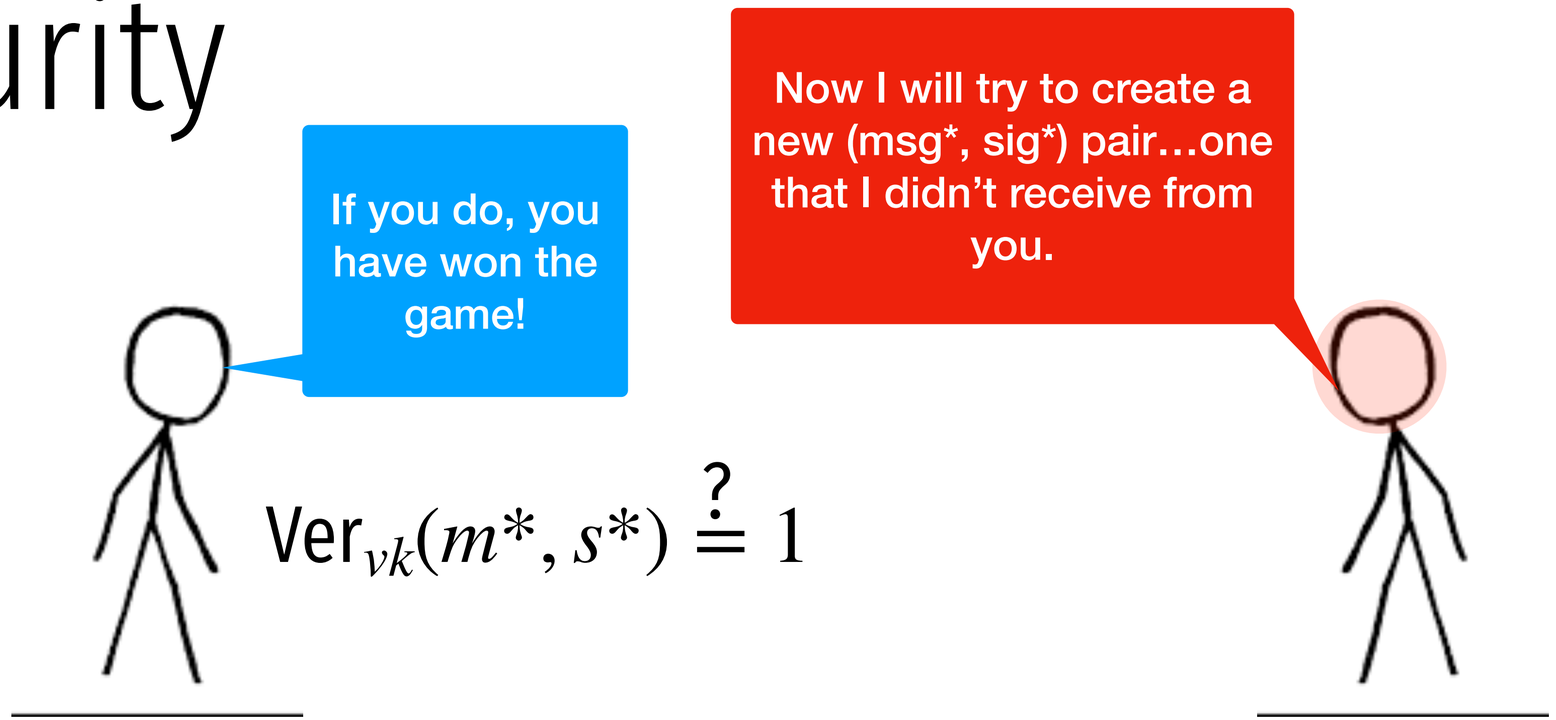
$(vk, sk) \leftarrow \text{Gen}(1^n)$

vk

$s_i \leftarrow \text{Sign}_{sk}(m_i)$

$s_1, s_2, \ldots$

Signature security



Captures the property of "unforgettability" of a new signature, even if the adversary can ask for arbitrary signatures of its choosing.

Textbook RSA Signatures (insecure)

Pick N = p*q where p,q are primes.

Pick e,d such that $e \cdot d = 1 \bmod \phi(N)$

Sign((sk=d, N) m):

Compute the signature: $\sigma \leftarrow m^d \bmod N$

Verify((pk=e, N), σ, m):

$$m \stackrel{?}{=} \sigma^e \bmod N$$

RSA Signatures in GPG

Sign((sk, N) m):

Compute the padding: $z \leftarrow 00 \cdot 01 \cdot FF \cdots FF \cdot 00 \cdot \mathsf{ID}_H \cdot H(m)$

Compute the signature: $\sigma \leftarrow z^{sk} \bmod N$

Example: apple.com

www.apple.com | Apple Public EV Server RSA CA 2 - G1 | DigiCert High Assurance EV Root CA

Subject Name

Business Category	Private Organization
Inc. Country	US
Inc. State/Province	California
Serial Number	C0806592
Country	US
State/Province	California
Locality	Cupertino
Organization	Apple Inc.
Common Name	www.apple.com

Issuer Name

Country	US
Organization	Apple Inc.
Common Name	Apple Public EV Server RSA CA 2 - G1

Validity

Not Before	Fri, 23 Aug 2024 17:30:11 GMT
Not After	Thu, 21 Nov 2024 17:40:11 GMT

Subject Alt Names

DNS Name	www.apple.com
DNS Name	images.apple.com
DNS Name	www.apple.com.cn

Public Key Info

Algorithm	RSA
Key Size	2048
Exponent	65537
Modulus	C8:A2:02:8A:C1:1C:A7:9A:EE:58:49:9B:10:3C:41:8D:BF:EF:6F:23:7E:64:05...

Safari is using an encrypted connection to www.apple.com.

Encryption with a digital certificate keeps information private as it's sent to or from the https website www.apple.com.

DigiCert, Inc. has identified www.apple.com as being owned by Apple Inc. in Cupertino, California, US.

DigiCert High Assurance EV Root CA
└ DigiCert SHA2 Extended Validation Server CA-3
└ www.apple.com

Serial Number	03 8E 3F 9E 09 D7 ED C7 B1 80 3F 74 A7 4C 35 AB
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters	None
Not Valid Before	Tuesday, October 6, 2020 at 8:00:00 PM Eastern Daylight Time
Not Valid After	Friday, October 8, 2021 at 8:00:00 AM Eastern Daylight Time
Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Parameters	None
Public Key	256 bytes : CA 1B 1C 21 78 15 3D 40 CF A3 79 3F 9D CF B2 53 AB A9 41 FF 3E 06 A1 29 69 8A 04 46 9E FB C4 0D 56 7A CA E6 80 E7 AF C6 C0 BF 8B 60 71 CA 9A E8 76 0C 06 C8 9B 77 B8 F3 1B EA 7E E7 3A 84 CB A3 88 A5 93 04 3F 69 66 77 CF AE 06 D1 D9 E1 10 08 7A E0 24 98 E7 56 97 0F 73 68 7B 4D 69 46 28 26 FF 05 81 0C C0 DA FC 21 71 81 65 9A 39 C9 E9 68 36 36 02 5F 81 80 B7 7E 8A 5B FE 34 D0 CE 76 2D D9 8B 3E D4 13 C0 EC EB 0F 2C 77 AD 1E 7B 20 F6 DA 92 98 FD 89 F3 A7 CB 53 16 2E B0 B9 62 BE C8 C3 28 40 CF 8C 5C 61 77 8F 92 3D 2F 23 F2 0A AB 65 82 22 B8 98 CE BA C8 00 95 E4 67 34 6E 76 E5 D1 D3 2D 51 91 BC EF C0 C8 DE F8 7B CC 46 45 00 76 D9 CB 30 31 E9 56 FD 0E 68 F4 36 F9 1B 5F 88 61 62 8F 60 A8 DE 43 7B 5C C1 15 73 D4 06 12 6E 85 9B 50 9C 24 BF 5F FC F4 68 95 67 D5 BF 44 71
Exponent	65537
Key Size	2,048 bits

? Hide Certificate OK

ECDSA Signatures

Works in an elliptic-curve group with generator G of prime order n (P256)

Secret key: random d Public key: $Q = d \cdot G$

Sign((sk = d), m):

Pick fresh random k: $(x_1, y_1) \leftarrow k \cdot G, \quad r \leftarrow x_1 \bmod n$

Compute: $s \leftarrow k^{-1}(H(m) + r \cdot d) \bmod n$ Output $\sigma = (r, s)$

Verify((pk = Q), σ, m):

Compute: $(x_1, y_1) \leftarrow H(m) \cdot s^{-1} \cdot G + r \cdot s^{-1} \cdot Q$

Accept iff $r \stackrel{?}{=} x_1 \bmod n$

Never reuse k: two signatures with the same k reveal d (how Sony's PS3 key leaked, 2010)

The specter of Quantum

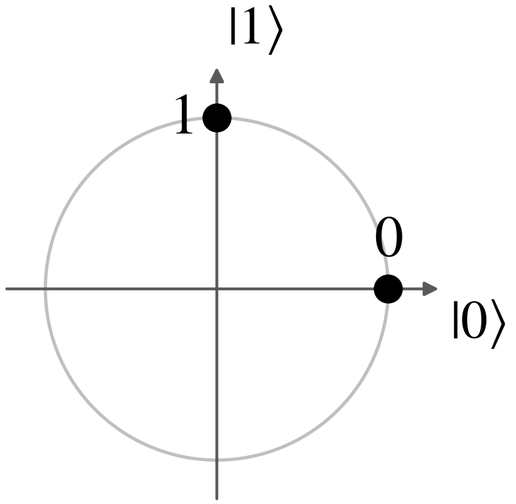
Bits vs. Qubits

Classical bit

Either 0 or 1

Reading it doesn't change it

n bits: one of 2^n values at a time



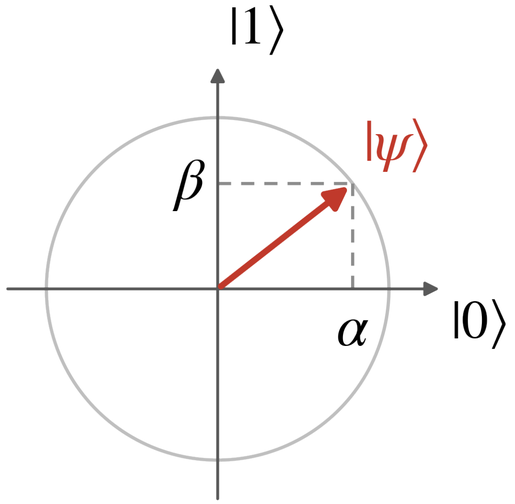
Qubit

$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad |\alpha|^2 + |\beta|^2 = 1$

Measuring: $\Pr[0] = |\alpha|^2, \quad \Pr[1] = |\beta|^2$

...and the superposition is destroyed

n qubits: 2^n amplitudes at once

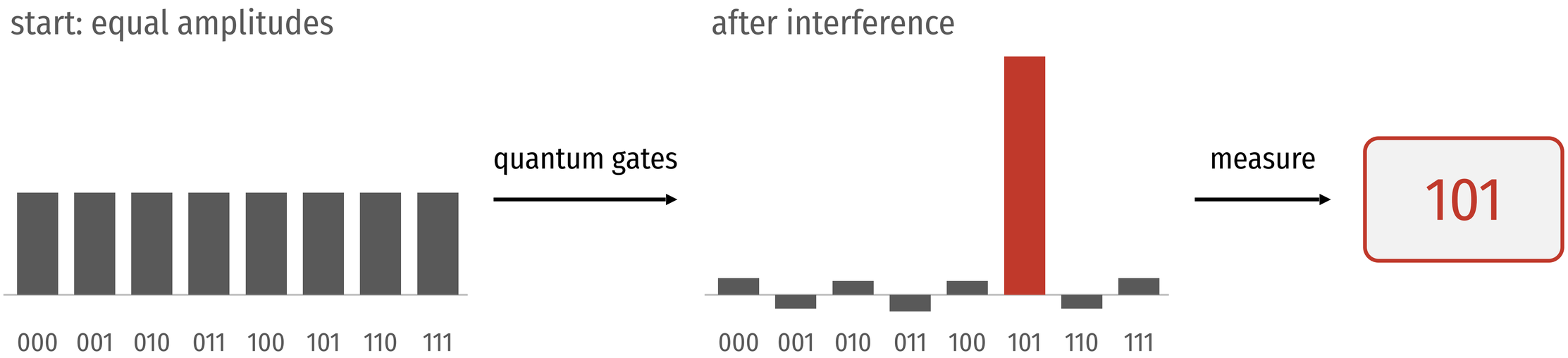


How Quantum Algorithms Work

Gates transform all 2^n amplitudes at once

But measuring gives only one outcome, at random

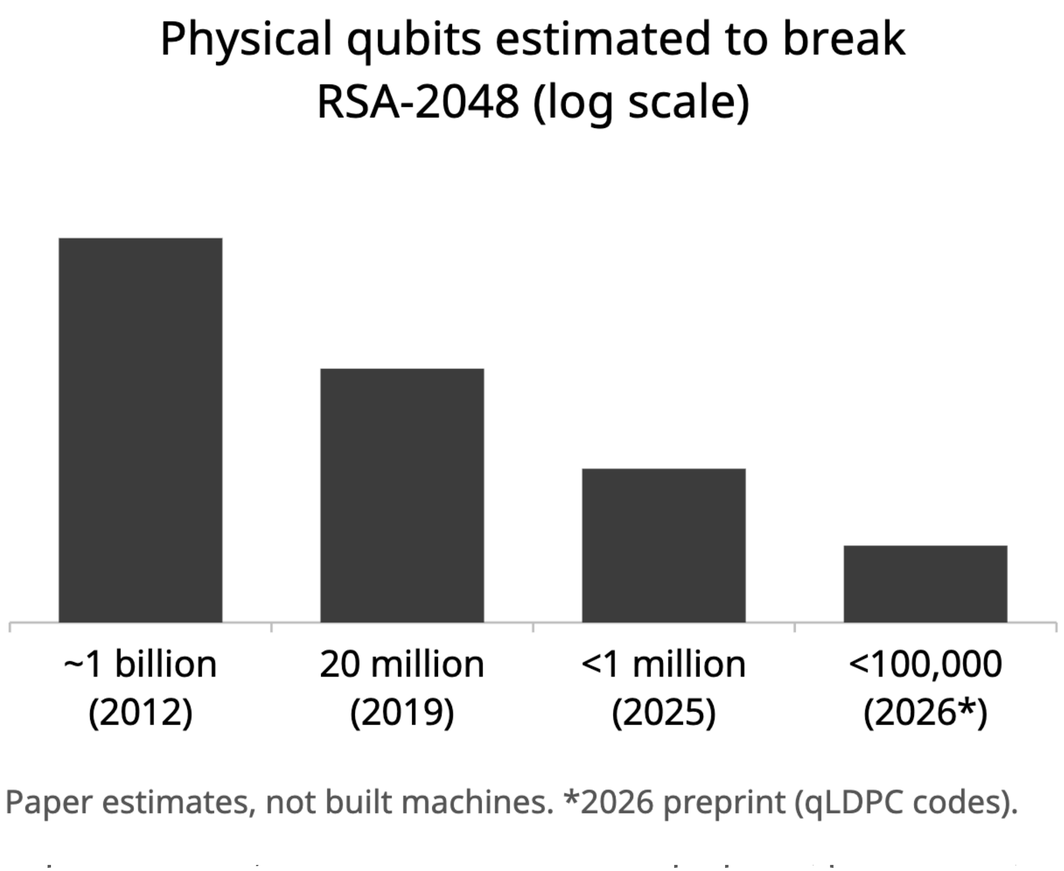
The trick is interference: wrong answers cancel, the right one adds up



Big speedups need hidden structure: factoring and discrete log (exponential, Shor).

Unstructured search: only a square-root speedup (Grover). Most problems: no big gain.

How Close Is a Quantum Attack?



Today: thousands of noisy physical qubits at most, few error-corrected ones. Shor has only factored toy numbers like 15 and 21.

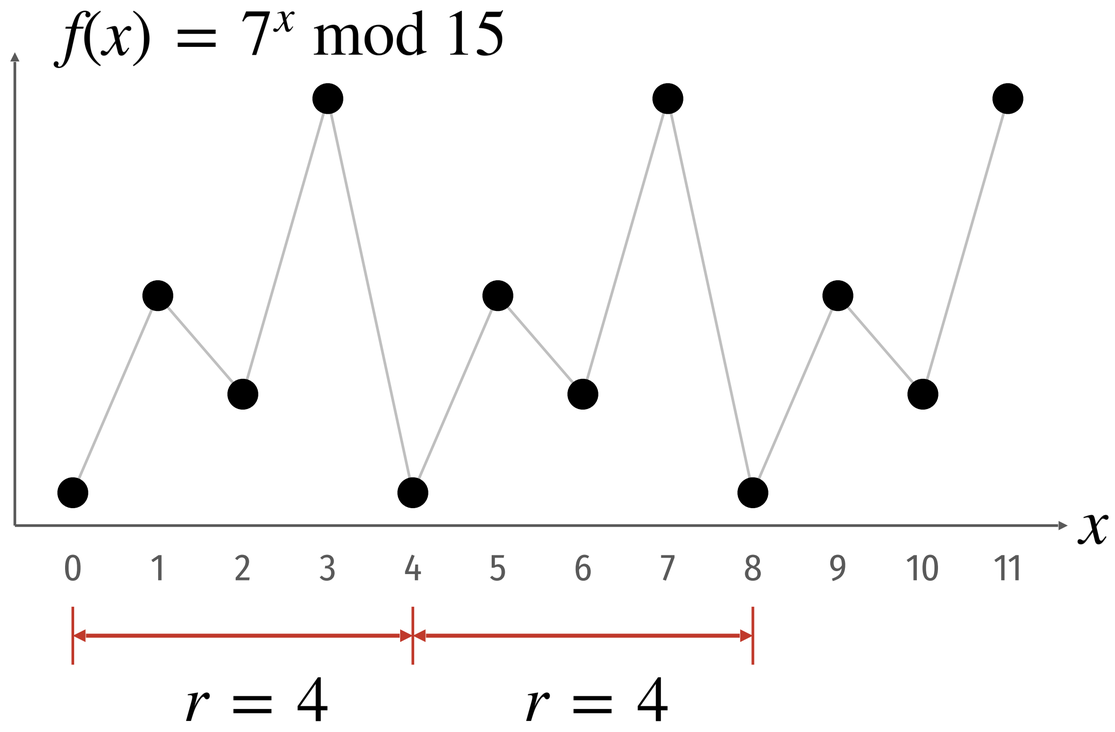
ECC may fall first: Google (2026) estimates a 256-bit elliptic-curve key could be broken in minutes with under 500,000 qubits.

Harvest now, decrypt later: traffic recorded today can be decrypted once a big machine exists, so encryption has to switch now.

Shor's Algorithm: Period Finding

Problem: find the hidden period r of a function f:

$f(x) = f(x + r)$ for every x



Classically: no fast method known (as hard as factoring)

Quantum (Shor, 1994):

1. Superpose all x, compute f(x)

2. Quantum Fourier Transform: amplitudes pile up on multiples of 1/r

3. Measure, then recover r classically

Polynomial: ~n^3 gates for an n-bit N

Shor Breaks RSA (Factoring)

1. Pick random a and let $f(x) = a^x \bmod N$
2. Quantum: find its period r, the smallest r with $a^r \equiv 1 \bmod N$
3. If r is even: $(a^{r/2} - 1)(a^{r/2} + 1) \equiv 0 \bmod N$

   so $\gcd(a^{r/2} - 1,\ N)$ is a factor p or q (with good probability)
4. With p and q: $\phi(N) = (p-1)(q-1), \quad d = e^{-1} \bmod \phi(N)$

   The attacker now has your RSA secret key: decrypt and forge at will

Example: N = 15, a = 7

$7^x \bmod 15 = 1, 7, 4, 13, 1, 7, 4, 13, \ldots$ so $r = 4$

$7^2 = 49 \equiv 4 \bmod 15$ so $\gcd(3, 15) = 3, \quad \gcd(5, 15) = 5$ 15 = 3 × 5 ✓

Shor Breaks ECDSA (Discrete Log)

ECDSA public key $Q = d \cdot G$: recovering d is a discrete log

Define $f(a, b) = a \cdot G + b \cdot Q$

Then $f(a + d,\ b - 1) = a \cdot G + b \cdot Q + (d \cdot G - Q) = f(a, b)$

So f has a hidden period $(d, -1)$

Shor finds the period, revealing d: now anyone can sign as you

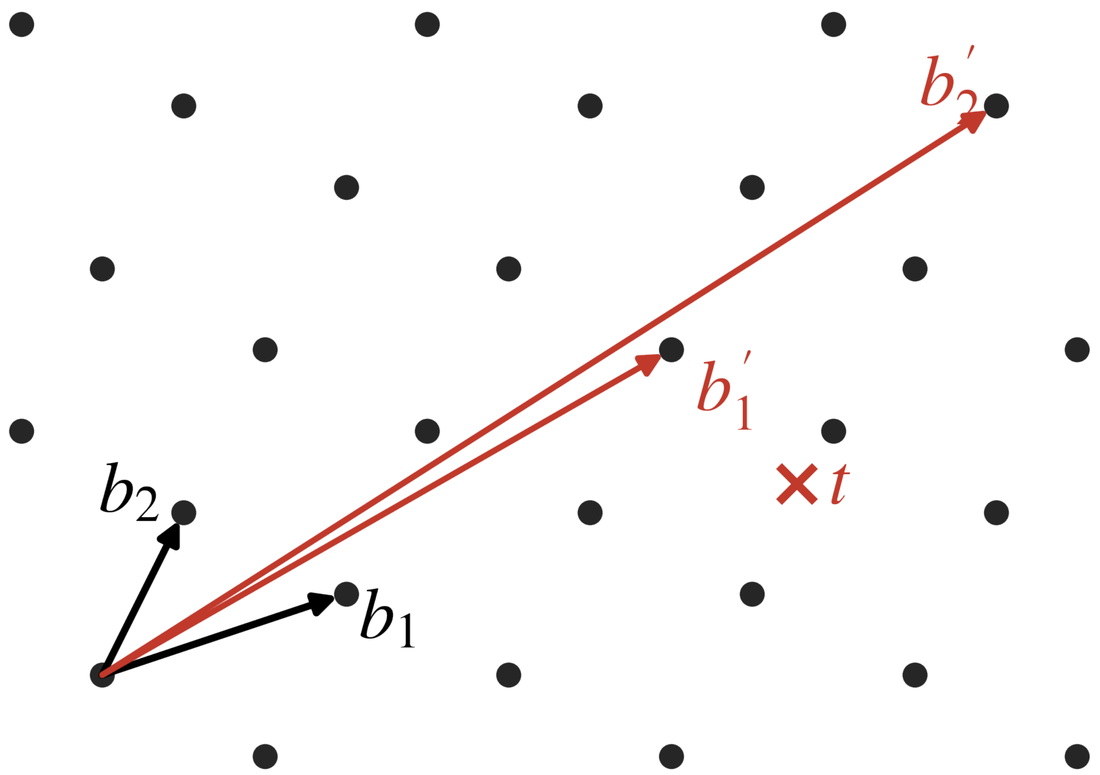
Broken by Shor

RSA, Diffie-Hellman, DSA, ECDH, ECDSA, EdDSA: essentially all public-key crypto deployed today

Survives (Grover: only a √ speedup)

AES, SHA-2, SHA-3: use 256-bit keys and outputs for a safety margin

Post-Quantum Crypto: Lattices



Same lattice, two bases: b_1, b_2 good (short), b_1', b_2' bad (long, skewed). Which point is closest to t?

A lattice: all integer combinations

$$L = \{ a\,b_1 + b\,b_2 \;:\; a, b \in \mathbb{Z} \}$$

Hard problems: find the shortest nonzero vector (SVP), or the lattice point closest to a target t (CVP)

Easy in 2-D with a good basis. With only a bad basis in ~1,000 dimensions: no fast algorithm known, classical or quantum

No hidden period for Shor to find

Trapdoor: a secret short basis (Falcon), or a secret hidden by noise (LWE, next)

Learning With Errors (LWE)

Public: modulus q and a random matrix $A \in \mathbb{Z}_q^{m \times n}$

KeyGen: secret s, small random noise e; publish $b = As + e \bmod q$

Enc((pk = A, b), bit μ):

Pick random $r \in \{0,1\}^m$ and compute

$u \leftarrow A^T r, \quad v \leftarrow r^T b + \mu \cdot \lfloor q/2 \rfloor$

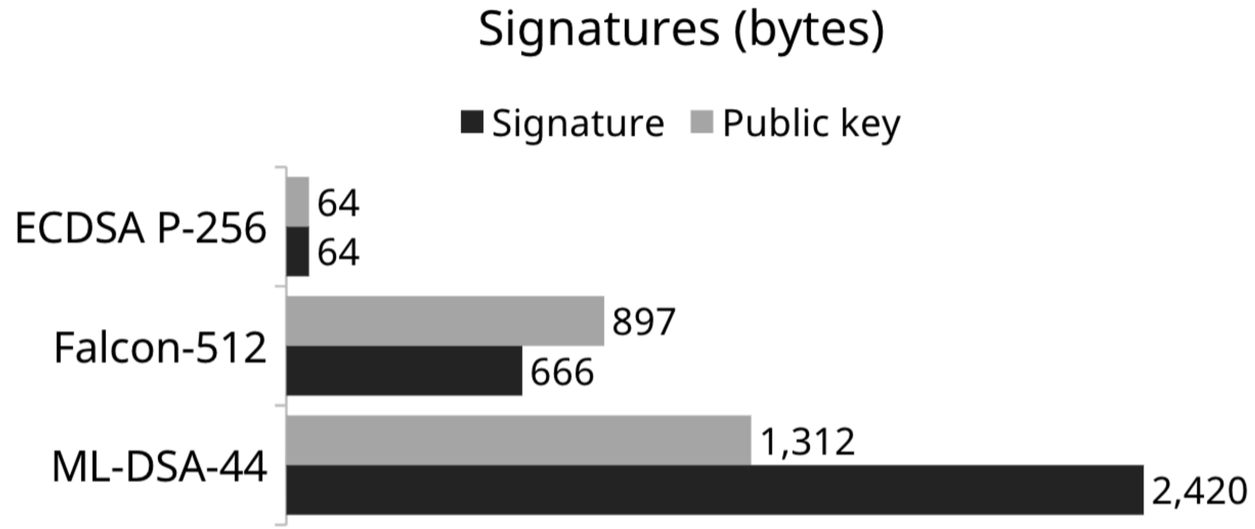
Dec((sk = s), (u, v)):

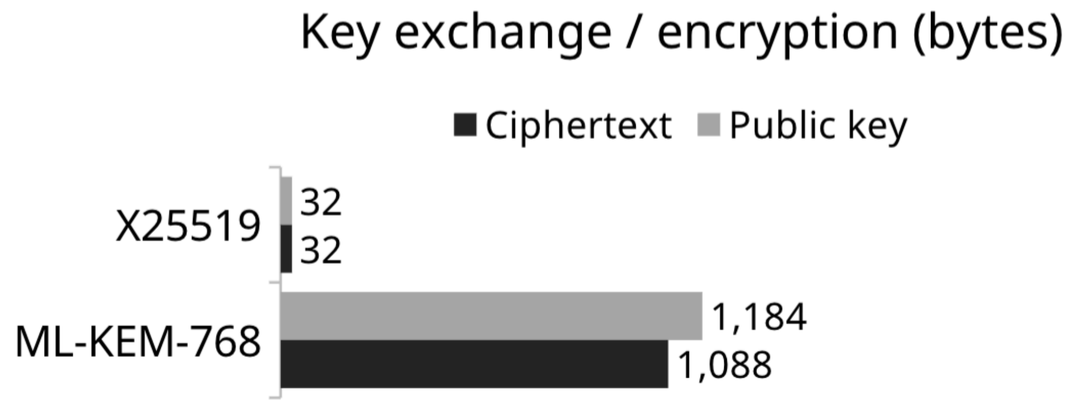
Compute $v - u^T s = r^T e + \mu \cdot \lfloor q/2 \rfloor$

$r^T e$ is small, so: near 0 → output 0, near q/2 → output 1

Without e, Gaussian elimination finds s. With e, it is as hard as worst-case lattice problems (Regev, 2005).
Standardized as ML-KEM (FIPS 203) and ML-DSA (FIPS 204), using structured “module” lattices.

Downsides of Lattice Crypto





ECDSA key and signature as raw (x, y) and (r, s) encodings

Bigger: ML-DSA signatures ~38× ECDSA; ML-KEM keys ~37× X25519. Heavier TLS handshakes and certificates.

Younger: LWE dates from 2005; factoring has been attacked for far longer. (SIKE, a non-lattice finalist, fell to a laptop in 2022.)

Tricky to implement: side channels, decryption failures; Falcon needs floating-point sampling.

So, deploy hybrids: X25519 + ML-KEM in TLS stays secure if either holds.