

In a public-key encryption scheme, Alice can send Bob secret messages over public channels without having exchanged any secret information in advance. The steps of public-key encryption are

1. **Key generation:** Bob runs the randomized key generation algorithm $Keys$ to produce a pair of keys: A secret key sk and a public key pk . He keeps sk to himself. He posts pk for anyone who is interested to use it.
2. **Encryption:** To communicate a message m , Alice runs encryption $Enc_{pk}(m)$ to compute the ciphertext c . She sends c to Bob.
3. **Decryption:** Upon receiving ciphertext c , Bob runs decryption $Dec_{sk}(c)$ to recover m .

An encryption scheme is functional if decryption inverts encryption: For every message m , and for every pair of keys (sk, pk) sampled by $Keys$, $Dec_{sk}(Enc_{pk}(m)) = m$.

We also want the encryption scheme to be secure. An eavesdropper Eve who observes the whole interaction should not be able to infer anything about the message. In private-key encryption, Eve sees only the encryption $Enc_{sk}(m)$ of Alice's message. Message indistinguishability requires that $Enc_{sk}(m)$ and $Enc_{sk}(m')$ are indistinguishable for every pair of messages m and m' . See Figure 1 (a).

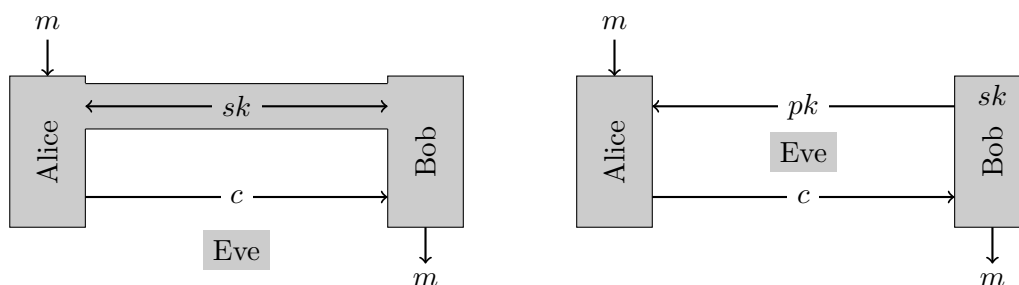


FIGURE 1: Private-key versus public-key encryption. (a) In a private-key scheme Alice and Bob have a shared secret key unknown to Eve. Eve observes Alice's ciphertext only. (b) In a public-key scheme Alice and Bob do not share any secrets prior to the interaction. Eve observes Bob's public key and Alice's ciphertext.

In public-key encryption, Eve also sees the public key pk . Message indistinguishability now posits that encryptions should be indistinguishable even given the public key (see Figure 1 (b)): For every two messages m and m' ,

$$(pk, Enc_{sk}(m)) \quad \text{is indistinguishable from} \quad (pk, Enc_{sk}(m'))$$

In the private-key setting, we were able to attain perfect indistinguishability as long as we were willing to put up with the key being very very long. For public-key encryption this is no longer true. Perfectly secure public-key encryption is impossible, regardless of key length. For Bob to be able to decrypt, different messages must always encrypt to different ciphertexts under the same public key. When $m \neq m'$, the random variables $(pk, Enc_{pk}(m))$ and $(pk, Enc_{pk}(m'))$ must have disjoint support. Eve can distinguish them perfectly if she is willing to put in the effort. Security has to rely on a gap in effort.

There is another important distinction. All we needed to build a reasonable private-key encryption was pseudorandomness. Pseudorandomness is certainly necessary for public-key encryption. An unstructured source of pseudorandomness is, however, no longer sufficient.

The first public-key encryption scheme was proposed by Whitfield Diffie and Martin Hellman in 1976. In 2015 they received the Turing Award for their invention. It took many years for them to be dutifully honored.



FIGURE 2: Martin Hellman and Whitfield Diffie. (From [Stanford](#).)

Two historical complications are likely to have contributed to this. The first is that Diffie and Hellman positioned their invention as a *key exchange* mechanism, not as public-key encryption. We now know that there is no fundamental distinction between the two, but it took a few years for cryptographers to realize that. It was in 1984 that Taher El Gamal pointed out the use of the Diffie-Hellman mechanism as message-indistinguishable encryption.

The other reason might be that in the early 1970s, cryptographers James Ellis and Clifford Cocks already knew about public-key encryption, but didn't publicize their discovery. Their employer at the time was GCHQ, the UK Government Communications Headquarters.

1 Modular arithmetic and discrete logarithms

Diffie-Hellman encryption has to do with exponentiation in finite groups. Let's start with the integers modulo p , denoted by $\mathbb{Z}_p$, where p is a very large prime number. For example, the RFC 7917 standard suggests choosing p as the 2048-bit, or 617-digit prime `ffdhe2048` =

```
323170060713110073001535134778251633624880571334890751745884341392698068341362
100027920563626401646854585563579353308169288290230805734726252735547424612457
410262025279165729728627063003252634282131457669314142236542209411113486299916
574782680342305530863490506355577122191878903327295696961297438562417412362372
251973464026918557977679768230146253979330580152268587307611975324364674758554
607150438968449403661304976978128542959586595975670512838521327844685229255045
682728791137200989318739591433741758378260002780349731985520606075332341226032
54684088120031105907484281003994966956119696956248629032338072839127039
```

In arithmetic modulo p we can not only add, subtract, and multiply, but also divide, as long as we do not divide by zero. If we exclude zero, we obtain the multiplicative group $\mathbb{Z}_p^*$ of order (size) $p - 1$ in which division of any two numbers is well-defined. If you forgot how to divide in modular arithmetic, [this is a good time to revise](#).

Another sensible operation in modular arithmetic is exponentiation. Given a group element g and an integer x , the power g^x can be calculated by multiplying g with itself x times. In the Diffie-Hellman algorithm, the exponent x has the same order of magnitude as p . Repeating multiplication this many times is infeasible. There is then a much faster way to exponentiate: If x is even, first calculate g^2 then raise it to the $(x/2)$ -th power recursively. If x is odd, do the same with $x - 1$ and multiply the result by g . This fast exponentiation algorithm reduces the number of multiplications from x to $O(\log x)$, namely the length of the bit representation of x .

Example. Here is how 5^{14} modulo 17 is calculated using fast modular exponentiation:

$$5^{14} = 25^7 \equiv 8^7 \equiv 8 \cdot 8^6 = 8 \cdot 81^3 \equiv 8 \cdot 13^3 = 8 \cdot 13 \cdot 13^2 = 8 \cdot 13 \cdot 169 \equiv 8 \cdot 13 \cdot 16 = 104 \cdot 16 \equiv 2 \cdot 16 = 32 \equiv 15.$$

When we work with real numbers, the operation inverse to powering is the logarithm. By analogy, we call an integer x a *discrete logarithm* of y in base b modulo p if $b^x \bmod p = y$. Discrete logarithms do not behave as well as continuous logarithms. Some numbers may have multiple logarithms, and some may have none at all.

Let's take the modulus $p = 11$ as an example. Then 7 has no discrete logarithm in base 4. 9, on the other hand, has two: Both 4^3 and 4^8 equal 9 modulo 11. The reason behind them is this theorem:

Theorem 1. For every prime number p there exists a number g between 1 and $p-1$ such that the sequence $g^0, g^1, \dots, g^{p-2}$ covers every nonzero number modulo p exactly once.

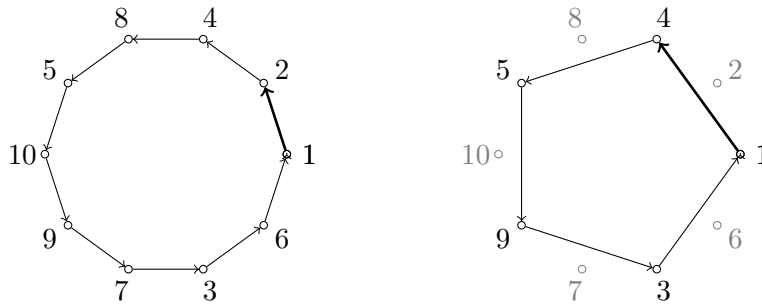


FIGURE 3: (a) $g = 2$ generates $\mathbb{Z}_{11}^*$. Arrows represent multiplication by 2. (b) $g^2 = 4$ generates the subgroup $\mathbb{G}$ of quadratic residues modulo 11. Arrows represent multiplication by 4.

Such a number g is called a *generator* of $\mathbb{Z}_p^*$. For example, $g = 2$ generates $\mathbb{Z}_{11}^*$. The sequence $1, 2, 2^2, \dots, 2^9$ covers every nonzero number modulo 11 exactly once; see Figure 3(a). The reason we stopped at 2^9 is because $2^{10} \equiv 1 \pmod{11}$. The result of 2^x modulo 11 only depends on the value of x modulo 10: $2^x \equiv 2^{x \bmod 10} \pmod{11}$. This is a special case of Fermat's little Theorem.

Fermat's little Theorem. If p is prime and h is nonzero modulo p then $h^{p-1} \equiv 1 \pmod{p}$.

Therefore $h^x \equiv h^{x \bmod (p-1)} \pmod{p}$. A discrete logarithm in modulus p can always be represented as a (nonzero) number modulo $p-1$. Theorem 1 then says that if the base g generates $\mathbb{Z}_p^*$ then every nonzero number modulo $p-1$ has a unique discrete logarithm in base g . The base-2 discrete logarithm of h modulo 11, for example, is the number of steps it takes to reach h in Figure 3(a).

If the base b does not generate $\mathbb{Z}_p^*$, then the powers of b will not cover all the nonzero numbers modulo p . Not every number will have a discrete logarithm in base b . For example, if $b = 4$ then the numbers that have discrete logarithms modulo 11 are the ones reachable from 1 in Figure 3(b). Now $4^5 \bmod 11 = 1$, so the value of the base-4 discrete logarithm, if it exists, reduces to a unique value between 0 and 4, and not between 0 and 9 as was the case in base 2.

This is no accident as $4 = 2^2$. The powers of 4 are the same as the even powers of 2 (modulo 11). The discrete logarithm can only exist for those even powers. As 2 generates $\mathbb{Z}_{11}^*$, these are precisely the squares, also known as *quadratic residues*. The quadratic residues modulo 11 are

$$1 \equiv 1^2 \quad 3 \equiv 5^2 \quad 4 \equiv 2^2 \quad 5 \equiv 4^2 \quad 9 \equiv 3^2.$$

The quadratic residues comprise exactly half of the numbers modulo 11. The product of quadratic residues is also a quadratic residue. In mathematical parlance, the quadratic residues form a order-5 subgroup $\mathbb{G}$ of $\mathbb{Z}_{11}^*$.

A prime number p is called a *safe prime* if $(p - 1)/2$ is also prime. Numbers 11 and `ffdhe2048` are examples of safe primes. The conclusions we drew about 11 are valid whenever the modulus p is a safe prime. If the base generates $\mathbb{Z}_p^*$, the discrete logarithm always exists and is unique modulo $p - 1$. If not, then the base generates the subgroup $\mathbb{G}$ of quadratic residues (and it is a quadratic residue itself). The discrete logarithm (of a quadratic residue) is then unique modulo $(p - 1)/2$.

2 The Diffie-Hellman assumptions

The Diffie-Hellman assumptions have to do with the intractability of calculating discrete logarithms. Unlike for exponentiation, we don't know of a general fast algorithm for this task.

The discrete log can be found by trying out all $p - 1$ possible values of x . Trying out 2^{2048} possibilities is infeasible. Somewhat more efficient algorithms do exist. Still, for `ffdhe2048` the best known algorithms require trying out more than 2^{100} possibilities. Calculating discrete logarithms modulo safe primes on the order of `ffdhe2048` is plausibly intractable.¹ This is the **computational Diffie-Hellman (CDH) assumption**.

The Decisional Diffie-Hellman (DDH) assumption postulates that discrete logarithms are not only difficult to find, but they are effectively indistinguishable from random numbers:

DDH assumption for base b :

Random variables (b^X, b^Y, b^{XY}) and (b^X, b^Y, b^Z) are indistinguishable with reasonable effort, where X, Y, Z are random exponents. All arithmetic is modulo p .

If discrete logarithms were easy to find, DDH cannot be true. For example, suppose Eve is challenged with the triple

$$(6\ 342\ 138, 9\ 728\ 259, 168\ 490). \quad (1)$$

The modulus is $p = 10\ 000\ 223$, which is a safe prime. The base is $b = 5$. Eve can distinguish between the two hypothesis by calculating the discrete logs of the three numbers. The modulus here is small enough that exhaustive search reveals the answers within a few seconds:

$$6\ 342\ 138 \equiv 5^{5\ 173\ 592} \quad 9\ 728\ 259 \equiv 5^{4\ 755\ 194} \quad 168\ 490 \equiv 5^{7\ 497\ 312} \quad \text{modulo } p$$

As

$$5\ 173\ 592 \cdot 4\ 755\ 194 \equiv 7\ 497\ 312 \pmod{p - 1} \quad (2)$$

this tuple is of the form (b^X, b^Y, b^{XY}) . It would have been extremely unlikely for (2) to happen by pure chance if the exponents were picked independently as in (b^X, b^Y, b^Z) . This method of distinguishing, however, is infeasible when p is on the order of `ffdhe2048`.

In this example, there is another distinguishing strategy that doesn't require calculating discrete logs. The base b happens to generate $\mathbb{Z}_p^*$. When X is a random exponent (modulo $p - 1$), b^X is equally likely to take any of the $p - 1$ values in $\mathbb{Z}_p^*$. The same is true for b^Y and for b^Z . It is, however, no longer true for b^{XY} .

The reason is that when X and Y are chosen at random modulo $p - 1$, at least one of them will be even with probability $3/4$. In that case, the exponent XY is even. The number b^{XY} is a quadratic residue with probability (at least) $3/4$.

Theorem 2. *If p is a safe prime and $h \not\equiv 0$, then $h^{(p-1)/2} \bmod p$ equals 1 if h is a quadratic residue and $p - 1$ if it isn't.*

¹If p breaks up into small prime factors this is no longer true.

For example, when $p = 11$, the quadratic residues are exactly those h for which $h^5 = 1$. These are precisely the numbers highlighted in Figure 3(b): $4^5 \equiv 1$, $5^5 \equiv 4^{10} \equiv 1$, $9^5 \equiv 4^{15} \equiv 1$, and so on. In contrast, $2^5 \equiv 10$, $8^5 \equiv 2^5 \cdot 4^5 \equiv 10 \cdot 1$, $10^5 \equiv 2^5 \cdot 5^5 \equiv 10 \cdot 1$, and so on.

Using Theorem 2, Eve can distinguish whether h is sampled as b^{XY} and b^Z . In the former case, $h^{(p-1)/2}$ takes value 1 with probability $3/4$. In the latter, the probability drops to half. In challenge (1), Eve would calculate $168\,490^{(p-1)/2} = 168\,490^{5\,000\,112}$ modulo $10\,000\,223$. The answer is 1. Eve is more likely to be looking at a Diffie-Hellman triple than at a random one.

This distinguishing strategy is effective even when p is very large. Eve only needs to perform a fast exponentiation. However, it no longer works if b itself is a quadratic residue.

To illustrate the last point, let's go back to $p = 11$. When b equals 2, b^Z is equally likely to take any value in $\mathbb{Z}_{11}^*$, while b^{XY} is quite a bit more likely to evaluate to one of the quadratic residues $\mathbb{G} = \{1, 3, 4, 5, 9\}$. When b equals 4, however, b^{XY} almost looks like a random quadratic residue. Its p.m.f. is

h	1	3	4	5	9
$\Pr(4^Z \equiv h \pmod{11})$	1/5	1/5	1/5	1/5	1/5
$\Pr(4^{XY} \equiv h \pmod{11})$	9/25	4/25	4/25	4/25	4/25

They are not quite identical, but they can effectively only be distinguished at value 1.² When p is `ffdhe2048`, the probability that b^{XY} takes value 1 is tiny. The random variables b^{XY} and b^Z are extremely close whenever b is a quadratic residue.

In summary, the DDH assumption is reasonable as long as b is a quadratic residue modulo p . As the relevant quantities b^X , b^Y , b^{XY} , and b^Z , are all quadratic residues, namely elements of $\mathbb{G}$, the assumption can be restated like this:

DDH assumption for quadratic residues modulo a safe prime:

For every quadratic residue $g \in \mathbb{G}$, (g^X, g^Y, g^{XY}) and (g^X, g^Y, g^Z) are indistinguishable with reasonable effort. X, Y, Z are random exponents modulo $(p-1)/2$, and all base arithmetic is modulo p .

3 Diffie-Hellman key exchange and El Gamal encryption

The DDH assumption gives a mechanism by which Alice and Bob can securely agree on a private key without prior interaction. This is the beautiful Diffie-Hellman key exchange. Fix a quadratic residue g .

Diffie-Hellman key exchange

- 1 Bob chooses a random X .
Alice chooses a random Y .
- 2 Bob sends g^X to Alice.
Alice sends g^Y to Bob.
- 3 Bob raises Alice's message to the X th power.
Alice raises Bob's message to the Y th power.

Alice and Bob both end up with $(g^X)^Y = (g^Y)^X = g^{XY}$, which they declare to be their secret key. What Eve sees from this interaction are the values g^X and g^Y . The DDH assumption says that she gains no information about the secret key g^{XY} because relative to g^X and g^Y , g^{XY} is indistinguishable from an independent random group element g^Z .

It is not difficult to turn this key exchange into a public-key encryption. Suppose Bob sends his message g^X before Alice does. Then Alice knows the secret key g^{XY} and can immediately use it as a one-time pad. Here is the resulting public-key encryption.

²The reason for the discrepancy is that $XY = 0$ when $X = 0$ or $Y = 0$. This is about twice as likely as $Z = 0$.

El Gamal encryption

Key generation: A random X is Bob's secret key. $pk = g^X$ is his public key.

Encryption: Message m is an element of $\mathbb{G}$. Alice encrypts it as $(g^Y, pk^Y \cdot m)$ with random Y .

Decryption: To decrypt (h, c) , Bob outputs $h^{-X} \cdot c$.

The scheme is functional. The decryption of $(g^Y, (g^X)^Y \cdot m)$ equals $(g^Y)^{-X} \cdot g^{XY} \cdot m = m$.

Security

El Gamal encryption is message indistinguishable under the DDH assumption. Eve observes the public key g^X , together with Alice's encryption $(g^Y, pk^Y \cdot m = g^Y, g^{XY} \cdot m)$. Together she sees the three group elements $(g^X, g^Y, g^{XY} \cdot m)$. If she can distinguish encryptions of m and m' , namely $(g^X, g^Y, g^{XY} \cdot m)$ from $(g^X, g^Y, g^{XY} \cdot m')$, then she can distinguish one of them, say the former, from (g^X, g^Y, g^Z) . In message indistinguishability, Eve chose m herself, so she can undo its effect by multiplying the third component with m^{-1} . Eve would then be effectively distinguishing (g^X, g^Y, g^{XY}) from $(g^X, g^Y, g^Z m^{-1})$. But $g^Z m^{-1}$ is identically distributed to g^Z even when conditioned on g^X, g^Y . So $(g^X, g^Y, g^Z m^{-1})$ is indistinguishable from (g^X, g^Y, g^Z) . Eve has therefore managed to distinguish (g^X, g^Y, g^{XY}) from (g^X, g^Y, g^Z) . She has broken the DDH assumption.

When we talked about private-key encryption, message indistinguishability was merely the stepping stone towards more realistic notions of security like multiple message indistinguishability and security against chosen plaintext attacks (CPA). We had to work somewhat hard to realize those enhanced security standards.

In public-key encryption, once Alice and Bob attain message indistinguishability, they get CPA security for free. Eve does not need to ask Alice to prepare encryptions of interest for her to examine. Eve knows the public key. She can sample those encryptions all by herself.

For related reasons, in public-key encryption, message indistinguishability is effectively oblivious to how many messages are exchanged. Suppose for example that Eve could distinguish an encryption of **make** followed by an encryption of **love**—let's denote this by **[make][love]**—from an encryption of **hate** followed by an encryption of **war**, namely **[hate][war]**. She can then distinguish one of them from **[make][war]**. Suppose for instance that she can tell **[hate][war]** from **[make][war]**. She can then tell **[hate]** from **[make]**: All she needs to do is tack on an encryption of **war**. She can create that one by herself as she knows Bob's public key.

An example and an annoyance

Let's illustrate the mechanics of El Gamal encryption in the group $\mathbb{G}$ of quadratic residue modulo the safe prime $p = 10\,000\,223$. The element $g = 4 = 2^2$ is a quadratic residue.

Bob picks a random exponent X between 0 and $(p-1)/2 = 5\,000\,112$ as his secret key, e.g., $X = 3\,428\,888$. His public key is $pk = g^X \bmod p = 8\,994\,707$.

Suppose Alice wants to encrypt $m = \text{she}$. Its ASCII encoding is $0x736865 = 7\,563\,365$. Alice picks a random pad Y between 0 and $(p-1)/2$, say $Y = 164\,304$. She calculates $g^Y \bmod p = 8\,788\,815$ and $pk^Y \cdot m \bmod p = 8\,788\,815$. She sends the ciphertext

$$(h = 8\,788\,815, c = 9\,327\,272)$$

to Bob.

To decrypt, Bob calculates $h^{-X} \cdot c$ modulo p . The multiplicative inverse h^{-1} of h is $6\,784\,579$, $h^{-X} = (h^{-1})^X$ is $7\,970\,197$, and $h^{-X} \cdot c$ is $7\,563\,365 = \text{she}$ as desired. (Alternatively, Bob could have used Fermat's little Theorem to calculate $h^{-X} \equiv h^{p-1-X} = 8\,788\,815^{6\,571\,334} \equiv 7\,970\,197$. This route avoids taking multiplicative inverses.)

Had Alice encrypted `her` = 0x686572 = 6 841 714 instead, the encryption would have been

$$(h = 8\,788\,815, c' = 2\,501\,045).$$

Ciphertexts (h, c) and (h', c') shouldn't be distinguishable to Eve, but they are. The reason is that $c = g^{XY} \cdot m$ is a quadratic residue, but $c' = g^{XY} \cdot m'$ is not. The difference comes from the message. The ASCII encoding of `she` happens to be a quadratic residue, but the one of `her` isn't.

For this reason the El Gamal algorithm requires that the message be a quadratic residue. Applying it directly to the numerical encoding of `her` was an improper use of the algorithm.

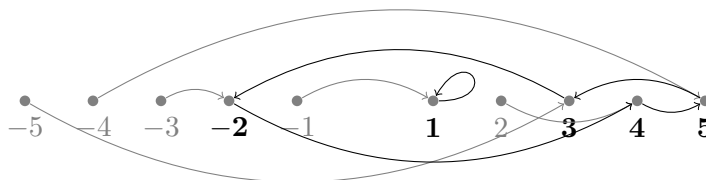


FIGURE 4: Powering in $\mathbb{Z}_{11}^*$. Arrows indicate squaring. The squares (in boldface) are the quadratic residues. Each has one positive and one negative square root.

The requirement that the message itself be a quadratic residue is annoying. Alice has no control over which plaintexts encode to quadratic residues and which don't. Luckily, there is a processing trick that ensures all plaintexts are represented by quadratic residues.

Instead of representing the group $\mathbb{Z}_p^*$ by the nonzero integers from 1 to $p - 1$, we'll use the numbers from $-q$ to q , where $q = (p - 1)/2$. Plaintexts will be represented by .

Plaintexts are represented by the positive numbers 1 to q .

Encoding: To encode a plaintext, square it to obtain a quadratic residue m .

Decoding: To decode m , raise it to the power $(q + 1)/2$ and take the absolute value.

For example, the ASCII encoding of plaintext `he` is 0x6865 = 26 725. The corresponding quadratic residue is $26\,725^2 \equiv 4\,209\,792$. This is the "message" m that Alice encrypts and that Bob decrypts. To convert it to its original representation, Bob calculates $m^{(q+1)/2} = 4\,209\,792^{2\,500\,056}$. This recovers the plaintext $26\,725 = \text{he}$.

Had Alice encoded `me` = 0x6d65 = 28 005 instead, she would have applied encryption to $28\,005^2 \equiv 4\,262\,631$. Bob would have decrypted to the same number. To recover the plaintext, Bob calculates $4\,262\,631^{2\,500\,056} \equiv 9\,972\,218 \equiv -28\,005$. Taking the absolute value recovers `me` again.

There is method in this madness. Every quadratic residue h has exactly two square roots. These are precisely $h^{(q+1)/2}$ and $-h^{(q+1)/2}$ modulo p . Among these two numbers, the one that is represented by h must be the positive one.

Figure 4 illustrates the mapping from plaintexts to quadratic residues modulo $p = 11$. In this example $q = (p - 1)/2 = 5$ and $(q + 1)/2 = 3$. To convert a plaintext between 1 and 5 to a quadratic residue h , square it. To go in the other direction, calculate h^3 and apply absolute value.

Here is why it works. If h is a quadratic residue, it has two square roots $\pm\sqrt{h}$. Then $h^3 = (\pm\sqrt{h})^6$. If $\sqrt{h}$ is the residue, $\sqrt{h}^5$ must be 1, so $h^3 = \sqrt{h}$. If $-\sqrt{h}$ is the residue, $(-\sqrt{h})^5$ must be 1, so $h^3 = -\sqrt{h}$. Either way, taking absolute value gives the unique positive square root of h .

4 RSA encryption

The Diffie-Hellman mechanism is the cleanest route to public-key encryption that I know. It has one practical drawback. Encryption entails modular powering with a large exponent. Even with fast modular

exponentiation, this operation takes up many cycles. In the 1970s this hindrance was significant. In 1978, Ron Rivest, Adi Shamir, and Len Adleman came up with their famous alternative encryption method.

Key Generation:

Sample two large distinct primes p and q at random. Set the modulus $n = pq$.

Choose public exponent e and secret exponent d so that $ed \equiv 1 \pmod{(p-1)(q-1)}$.

Publish n and e .

Encryption: m encrypts to $c = m^e \bmod n$.

Decryption: c decrypts to $c^d \bmod n$.

The functionality of RSA encryption comes from Euler's theorem: If x is coprime to p and q then $x^{(p-1)(q-1)} \equiv 1 \pmod{n}$. Therefore for some k ,

$$c^d = m^{ed} = m^{k(p-1)(q-1)+1} = (m^{(p-1)(q-1)})^k \cdot m \equiv 1 \cdot m = m$$

modulo n .

Example

Suppose Bob picks the prime numbers

$$p = 5003 \quad \text{and} \quad q = 5009$$

Their product is the modulus

$$n = pq = 25\,060\,027.$$

Bob then sets his public exponent e to 3. This is coprime to p and q . The product $(p-1)(q-1)$ equals 25 050 016. The master equation $3d \equiv 1 \pmod{25\,050\,016}$ solves to $d = 3^{-1} \bmod 25\,050\,016 = 16\,700\,011$. This is Bob's secret exponent.

Alice now wants to encrypt **her** = 6 841 714. The ciphertext is $c = \text{her}^3 \bmod n = 7\,241\,582$.

To decrypt, Bob calculates $7\,241\,582^{16\,700\,011} \bmod n = 6\,841\,714 = \text{her}$.

Security

RSA encryption is not message indistinguishable for a simple reason. Encryption is not randomized. Eve can distinguish encryptions of **love** from encryptions of **hate** because she can calculate them herself. The ciphertexts sent out by Alice will be exactly the same.

RSA plausibly satisfies a weaker notion of security. If Eve were to observe the encryption of a *random* message m , she should not be able to extract any "interesting" information about it.

The assumption that messages are random was problematic for private-key encryption. The reason was that Eve could possess side information. There is an important use of public-key encryption where random messages are a reasonable assumption. The application is key exchange. If m is a shared key to be used for further private-key encryption, Alice can choose it at encryption time. No side information will be leaked.

What counts as interesting information is not so easy to pin down. Even if m was random, Eve does learn its e -th power modulo n . Whether this is of interest or not depends on how m will further be used.

For these reasons, RSA encryption is not recommended for use in its original form. Later in the course we will see tools that can upgrade it into a truly secure scheme.

Any security claim goes out the window if Eve is somehow able to decompose n into its prime factors p and q . She can then calculate the exponent modulus $(p-1)(q-1)$ and recover the private key by solving the equation $ed \equiv 1 \pmod{(p-1)(q-1)}$. Hardness of factoring products of large primes is thus necessary for security. It is not known if it is sufficient. All proposed attacks on RSA so far, however, have tried to factor n first. If n is 2000 to 4000 bits long, as in recommended standards, this is believed to be infeasible.

Implementation

The highlight of RSA is its efficient implementation. This is why it became the workhorse of internet commerce. The internet as we know it would not have been the same without it. The 2002 Turing Award honors its inventors “for their seminal contributions to the theory and practical application of public-key cryptography.”



FIGURE 5: Ron Rivest, Adi Shamir, and Len Adleman.

In El Gamal encryption, the modulus p was fixed once and for all. In RSA, the primes p and q are picked at key generation time. Different runs of the key generation algorithm should produce different primes and different moduli. An important component of RSA key generation is sampling a random prime number.

Primality testing The only known method for uniformly sampling a large prime number is to sample a large random number and hope that it is prime. The probability that a random n -bit number happens to be prime is about $1/n \ln 2 \approx 1.443/n$. When n is 2000-bits long this is a bit less than $1/1000$. Out of 1000 trials, one number will turn out to be prime on average.

Once a candidate prime number has been sampled, it has to be tested for primality. Algorithms for primality testing are somewhat complicated. The first and possibly simplest such algorithm was invented by Gary Miller in 1976 and simplified by Michael Rabin in 1980. The idea is to use Fermat’s little theorem as a primality criterion. To test if p is prime, Bob chooses some base b and calculates b^{p-1} modulo p . If p was prime, Bob should always obtain 1.

What if p was composite? The mathematics is a little complicated. For many composite numbers p , b^{p-1} modulo p is not one for at least half of the possible bases b . Bob can establish great confidence that such numbers are composite by trying out, say, 50 random values of b . If p was one of these composite numbers, the chances of picking 50 misleading xs are extremely small.

The complication is that there are exceptions called Carmichael Numbers. Carmichael numbers p are composite, yet $b^{p-1} \equiv 1$ for all bases b . One example is $561 = 3 \cdot 11 \cdot 17$. There is, however, a stronger property that is only true for prime numbers. As b^{p-1} is always 1 when p is prime, its square root $b^{(p-1)/2}$ must be either 1 or -1 . If it is 1, its own square root $b^{(p-1)/4}$ must be either 1 or -1 , assuming $(p-1)/2$ is even. Chipping off factors of two in the exponent, one at a time, one is assured to get a sequence of 1s, until either a -1 is encountered, or the exponent becomes odd. It turns out that prime numbers are truly unique in this way. If p was composite (but not the power of another prime), the sequence $b^{p-1}, b^{(p-1)/2}, b^{(p-1)/4}, \dots$ looks different for at least half of the bs .

The encryption exponent The main advantage of RSA is the flexibility in choosing the encryption exponent e . Unlike p , q , and d , e can be set in stone, once and for all. The only essential requirement is that e not be coprime to both $p-1$ and $q-1$ so that Bob can solve the master equation $ed \equiv 1 \pmod{(p-1)(q-1)}$ for d . Popular choices of e are $17 = 2^4 + 1$, and $65\,537 = 2^{16} + 1$. These choices result in extremely fast modular exponentiation.

There is some price in security to be paid when the exponent is small. The reason is that if m also happens to be small, the reduction of m^e modulo n might not kick in. m^e modulo n would then be the same as m^e sans modulus. Eve can extract the message by taking the integer e -th root.

Decryption The decryption exponent d is typically large. Decryption $c^d \bmod n$ takes longer than encryption. There is a way to speed up the calculation. The **Chinese Remainder Theorem** says that $u = c^d \bmod p$ and $v = c^d \bmod q$ determine $c^d \bmod n$. As p and q have roughly half the bitsize of n , u and v can be calculated somewhat faster. The message $m = c^d$ can be recovered using the formula $m = (u \cdot (q^{-1} \bmod p) \cdot q + v \cdot (p^{-1} \bmod q) \cdot p) \bmod n$.

In the above example, $p^{-1} \bmod q = 5003^{-1} \bmod 5009 = 4174$ and $q^{-1} \bmod p = 5009^{-1} \bmod 5003 = 834$. The ciphertext is $c = 7\,241\,582$. As $c \bmod p = 2241$ and $d \bmod (p-1) = 3335$, $u = 2241^{3335} \bmod p = 2613$. Similarly, $c \bmod q = 3577$, $d \bmod (q-1) = 3339$, and $v = 3577^{3339} \bmod q = 4429$. Finally, $m = (2613 \cdot 834 \cdot 5009 + 4429 \cdot 4174 \cdot 5003) \bmod n = 6\,841\,714 = \text{her}$.

5 Learning with Errors (optional)

In 1994 Peter Shor demonstrated that discrete logarithms and prime factors can be found quickly on a quantum computer. If fault-tolerant quantum computers pan out, Diffie-Hellman key exchange, El Gamal encryption, and RSA encryption would be instantly broken. A third type of public-key encryption, however, would plausibly still remain secure. The ideas originate in work of Miklós Ajtai from the 1990s.

The Learning with Errors (LWE) assumption is about the hardness of solving noisy linear equations in rings. The domain is the ring $\mathbb{Z}_q$ of integers modulo q . There are several variants of this assumption which turn out to be roughly the same.

To create a *solvable* system of n linear equations in n variables modulo q , a secret solution $\mathbf{x}$ is first chosen. Equations are sampled at random in a way that is consistent with the solution. For example, if $n = 3$ and $q = 191$ the system may look like this:

$$\begin{aligned} 143x_1 + 156x_2 + 161x_3 &= 153 \\ 156x_1 + 30x_2 + 33x_3 &= 188 \\ 150x_1 + 24x_2 + 154x_3 &= 87 \end{aligned}$$

The “secret” $\mathbf{x}$ can be found using modular Gaussian elimination. In this example the solution is $\mathbf{x} = (3, -1, 4)$. We will insist that the solution $\mathbf{x}$ is *b-short*: The numbers $x_1, \dots, x_n$ are between $-b$ and b modulo q , with b much smaller than q . In our example $b = 5$. Things change dramatically if noise values in the range $-b$ to b are added to the right-hand side:

$$\begin{aligned} 143x_1 + 156x_2 + 161x_3 &= 151 \pm 5 \\ 156x_1 + 30x_2 + 33x_3 &= 188 \pm 5 \\ 215x_1 + 138x_2 + 49x_3 &= 92 \pm 5. \end{aligned}$$

A solution $\mathbf{x}$ can still be found by applying Gaussian elimination to the system with the corrupted right-hand side. In this example, if we ignore the ± 5 on the right, the unique solution is $(59, 41, 31)$. This solution is not short. Finding a solution that is short seems to be much more difficult. We know that one must exist because it was planted in there.

Search-LWE assumption: Given a random square system of approximate modular equations $A\mathbf{x} \approx \mathbf{y}$ with a planted short solution $\mathbf{x}$, finding this solution, or any other short solution, is intractable.

The right-hand size $\mathbf{y}$ is itself of the form $A\mathbf{x} + \mathbf{e}$, where $\mathbf{x}$ is the planted solution and $\mathbf{e}$ is the corruption noise. In a typical instantiation, the system might have several hundred equations and unknowns, and the modulus q might be in the thousands.

Just like in the case of DDH, for designing public-key encryption a decisional assumption is more useful. The (decisional) LWE assumption states that the right-hand side of the noisy system of equations is indistinguishable from random.

Learning with errors (LWE) assumption: The random variables $(A, \mathbf{y})$ and $(A, \mathbf{r})$, where $\mathbf{r}$ is a random n -dimensional vector modulo q , are indistinguishable with reasonable effort.



FIGURE 6: Miklós Ajtai.

Key exchange

There are several algorithms for public-key encryption whose security is based on the LWE assumption. Here is a key exchange algorithm.

Algorithm *DXL* (Ding-Xie-Lin approximate key exchange)

- 1 Alice samples a random short row vector $\mathbf{x}'$.
Bob samples a random short column vector $\mathbf{x}$.
- 2 Alice and Bob sample a uniformly random n by n matrix A .
- 3 Bob sends $A\mathbf{x} + \mathbf{e}$ to Alice, with a random b -short $\mathbf{e}$.
Alice sends $\mathbf{e}' + \mathbf{x}'A$ to Bob, with a random b -short $\mathbf{e}'$.
- 4 Alice multiplies Bob's message by $\mathbf{x}'$ on the left to produce her secret key.
Bob multiplies Alice's message by $\mathbf{x}$ on the right to produce his secret key.

Just like Diffie-Hellman key exchange led to El Gamal encryption, Ding-Xie-Lin can be converted into an LWE-based public-key encryption scheme. (Historically, Oded Regev invented the encryption first.)

Alice obtains the value $\mathbf{x}'(A\mathbf{x} + \mathbf{e}) = \mathbf{x}'A\mathbf{x} + \mathbf{x}'\mathbf{e}$. Bob obtains the value $(\mathbf{e}' + \mathbf{x}'A)\mathbf{x} = \mathbf{e}'\mathbf{x} + \mathbf{x}'A\mathbf{x}$. As A is a random matrix, both of them are very close to uniformly random numbers modulo q .³ But they are not exactly the same.

This is where the LWE assumption kicks in. As $\mathbf{x}$, $\mathbf{x}'$, $\mathbf{e}$, and $\mathbf{e}'$ are b -bounded, the dot products $\mathbf{x}'\mathbf{e}$ and $\mathbf{e}'\mathbf{x}$ have magnitude at most b^2n . If q is sufficiently larger than b^2n , Alice's and Bob's keys are still very close to one another relative to their magnitude:

$$\mathbf{x}'(A\mathbf{x} + \mathbf{e}) \approx \mathbf{x}'A\mathbf{x} \approx (\mathbf{e}' + \mathbf{x}'A)\mathbf{x}.$$

The small discrepancy in Alice's and Bob's keys does not affect the utility of key exchange.

How about security? Eve observes the messages $(A, A\mathbf{x} + \mathbf{e}, \mathbf{e}' + \mathbf{x}'A)$. By the LWE assumption, both $A\mathbf{x} + \mathbf{e}$ and $\mathbf{e}' + \mathbf{x}'A$ appear independent of A . What Eve sees looks indistinguishable from random numbers modulo q . For security of key exchange, this is not sufficient. What Eve sees should also reveal

³Some error is incurred by the event $\mathbf{x} = 0$ or $\mathbf{x}' = 0$, but this is extremely rare.

no information about Alice's and Bob's derived keys.⁴ This is still true, but it takes some extra work to complete the reasoning.

6 Summary

Public-key cryptography was proposed by Diffie and Hellman in 1976. The promise of secret communication between two people who have never met was revolutionary. Their work was not motivated by practical needs. Wide-area networks like the internet had to wait for another fifteen years. Once long-distance communication over infrastructure beyond anyone's control became a reality, public-key encryption proved indispensable. It is difficult to imagine how e-commerce would operate without it.

A generic source of pseudorandomness was sufficient to obtain private-key encryption. This is no longer true for public-key encryption. The security of public-key schemes rests on specific assumptions such as intractability of discrete logarithms, factoring large numbers, or solving noisy modular linear equations.

Public-key encryption algorithms are much less efficient than private-key ones. Their running time is larger by thousands of times. For this reason they are often used for *hybrid encryption*. The message in the public-key scheme is itself a random secret key. Once Bob receives it, Alice and Bob can use it as the shared key for private-key encryption. RSA was specifically designed for this purpose. It does not satisfy usual notions of security like message indistinguishability.

It is believed that quantum computers, if ever built, will have the ability to quickly take discrete logarithms and factor large numbers. Encryption schemes that rely on the intractability of these problems would be immediately broken. On the other hand, quantum computers are not known to provide substantial advantage in solving noisy linear equations. To prepare for this eventual post-quantum future, proposals based on the Learning with Errors were standardized in the early 2020s.

⁴Owing to the small disagreement, the two are not identical.