6 Public Key Cryptography

Last week in section 5 we discussed a public key-exchange algorithm, in which two parties can securely exchange an encryption key over an insecure connection, so that they have access to the same key but an evesdropper does not.

This week we will study an even more significant advance: the ability to do encryption without a shared secret key at all.

A public-key cryptosystem requires the generation of two paired keys: the *private key* and the *public key*. Alice generates a public-private key pair (k_{pub}, k_{priv}) and publishes k_{pub}. When Bob wants to send a message to Alice, he encrypts it with k_{pub} and transmits it; it can only be decrypted with k_{priv}, which only Alice has access to.

If Alice wants to send a message to Bob, she needs him to generate his own keypair and publish his own public key. This leads to the practice among many seriously security-minded internet users of listing a public key on their website for authentication and encrypted communications.

6.1 The order of an integer

We first need to recall and extend a couple of last week's number theory results about primitive roots and exponentiation in modular arithmetic.

Theorem 6.1 (Fermat's Little Theorem)**.** *Let p be a prime. If $\gcd(a,p) = 1$, then $a^{p-1} \equiv 1 \mod p$.*

Sometimes, we instead say that $a^p \equiv a \mod p$. These two statements are equivalent.

Remark 6.2. This theorem was first proven by Leonhard Euler.

Definition 6.3. Let m be a positive integer. We define the *Euler totient function* $\phi(m)$ to be the number of integers between 0 and m that are relatively prime to m.

There is a straightforward way to compute $\phi(m)$, but it's a bit too complicated to explain here. We will state the limited result that if p, q are primes, then $\phi(p) = p - 1$ and $\phi(pq) = (p-1)(q-1)$, which we will need.

Theorem 6.4 (Euler's Theorem)**.** *If a, m are natural numbers and $\gcd(a,n) = 1$, then $a^{\phi(m)} \equiv 1 \mod m$.*

Example 6.5. Let $p = 7$ and $q = 11$. We can see that, for instance,

$$\begin{aligned} 3^6 &= 3^3 \cdot 3^3 \equiv (-1)(-1) \equiv 1 \mod 7 \\ 3^{10} &= (3^2)^5 \equiv (-2)^5 \equiv -32 \equiv 1 \mod 11 \\ 3^{60} &= (3^4)^{15} \equiv 4^{15} \equiv (4^3)^5 \equiv (-13)^5 \equiv -13(13^2)^2 \equiv -13(15)^2 \\ &\equiv -39 \cdot 75 \equiv -39 \cdot (-2) \equiv 1 \mod 77 \end{aligned}$$

Remark 6.6. Fermat's little theorem is a special case of Euler's theorem, and follows from the fact that if p is prime then $\phi(p) = p - 1$.

Euler's theorem implies that for any number a, there is at least one integer r such that $a^r \equiv 1 \mod m$. This allows us to give the following definition:

Definition 6.7. Let a, m be integers, and suppose $\gcd(a, m) = 1$. Then the *order* of a mod m, written $\text{ord}_m(a)$, is the smallest positive integer r such that $a^r \equiv 1 \mod m$.

We observe that $1 \leq \text{ord}_m(a) \leq \phi(m)$ for any a.

Example 6.8. $\text{ord}_7(2) = 3$ since $2^3 = 8 \equiv 1$, and no smaller number works.

$\text{ord}_7(3) = 6$ since that's the smallest power of 3 that gives us 1; we have $3, 2, 6, 4, 5, 1$.

$\text{ord}_{10}(3) = 4$ since we compute $3, 9, 7, 1$.

Remark 6.9. A number g is a primitive root mod p if and only if $\text{ord}_p(g) = p - 1$. More generally, g is a primitive root mod m if and only if $\text{ord}_m(g) = \phi(m)$.

Fact 6.10. *$a^r \equiv a^s \mod m$ if and only if $r \equiv s \mod \text{ord}_m(a)$.*

In particular, if $r \equiv s \mod \phi(m)$ then $a^r \equiv a^s \mod m$.

6.2 The El-Gamal Cryptosystem

The El-Gamal Cryptosystem is a public-key cryptosystem originally described by the Egyptian mathematician Taher Elgamal in 1985. Its security relies on the difficulty of computing a discrete logarithm, and acts in a sense as an extension of the Diffie-Hellman process we discussed in section 5.

Algorithm 6.1 (El-Gamal Cryptosystem)**.** First Alice generates a private key and a public key.

1. Choose a large prime number p, and an element $g \in \mathbb{Z}/p\mathbb{Z}$ such that $\text{ord}_p(g)$ is a large prime number. This step is not at all trivial, and thus tends to be pre-standardized; everyone uses the same p and g.

2. Alice chooses a secret number a, which we call the *private key*. She does not share this number with anyone.

3. Alice computes $A \equiv g^a \mod p$, and publishes it. We call this number the *public key* because it is released to the public.

Now suppose Bob wishes to send Alice a number $2 < m < p$.

1. Bob generates a random number k, called the *ephemeral key*. This key is kept secret, and also discarded after this single message is sent.

2. Bob computes $c_1 \equiv g^k \mod p$ and $c_2 \equiv mA^k \mod p$. He sends Alice the message (c_1, c_2).

How does Alice decrypt the message? She has to use her private key a.

1. Alice computes $x \equiv c_1^a \mod p$, and then $x^{-1} \equiv c_1^{-a} \mod p$. (Alternately, she can just compute $x^{-1} \equiv c_1^{p-1-a} \mod p$ and skip computing x at all).

2. Alice then computes $c_2 x^{-1} \mod p$, which is equivalent to m.

Remark 6.11. The message is a number between 2 and m, and so takes $\log_2(m)$ bits to represent. The ciphertext consists of *two* integers between 2 and m, and thus takes $2\log_2(m)$ bits to represent. This the El-Gamal cryptosystem expands messages by a factor of two.

Remark 6.12. The requirement that $\text{ord}_p(g)$ is a large prime defeats a specific attack based on something called quadratic reciprocity, which tells us whether a number is a square modulo p.

This requirement is generally met by taking the large prime p to be of the form $2q + 1$ where q is also prime. We can check that any element either has order $1, 2, q$, or $2q$, and we want one with order q. Most choices will have order either q or $2q$.

If g is a primitive root mod p then $\text{ord}_p(g^2) = q$. But the easiest way to find an appropriate base g for the ElGamal algorithm is to simply choose a number and raise it to the q power; if this is equivalent to 1 mod p, we have what we're looking for.

Proposition 6.13. *The decryption step of Algorithm 6.1 works. That is,* $c_2 x^{-1} \equiv m \mod p$.

Proof.

$$\begin{aligned} c_2 x^{-1} &\equiv c_2 (c_1^a)^{-1} \\ &\equiv (mA^k)(g^{ak})^{-1} \\ &\equiv (mg^{ak})g^{-ak} \\ &\equiv mg^{ak}g^{-ak} \equiv m \mod p. \end{aligned}$$

□

Example 6.14. Suppose we take $p = 467$ and $g = 4$. Alice chooses $a = 155$ as her private key, and computes $A \equiv g^a \equiv 4^{155} \equiv 43 \mod 467$. (She can do this computation using the fast exponentiation algorithm from section 5.6). Alice publishes the number A.

Now suppose Bob wants to send the message $m = 42$ to Alice. Bob himself chooses an ephemeral key $k = 187$. He computes:

$$\begin{aligned} c_1 &\equiv g^k \equiv 4^{187} \equiv 456 \mod 467 \\ c_2 &\equiv mA^k \equiv 42 \cdot 43^{187} \equiv 67 \mod 467. \end{aligned}$$

Bob sends Alice the message $(456, 67)$.

Alice wishes to decrypt the message. She computes:

$$\begin{aligned} x &\equiv c_1^a \equiv 456^{155} \equiv 413 \mod 467 \\ x^{-1} &\equiv c_1^{p-1-a} \equiv 147 \mod 467 \\ m &\equiv c_2 x^{-1} \equiv 67 \cdot 147 \equiv 9849 \equiv 42 \mod 467. \end{aligned}$$

Remark 6.15. Note a very important property here: Bob will send a different ciphertext depending on his random choice of k, but Alice will decrypt it to the same message regardless of Bob's choice of k. This means that there are many different ciphertexts corresponding to the same plaintext; this is why the ciphertext (which is a pair of integers) has twice as many bits as the plaintext (which is a single integer).

6.2.1 Cryptanalysis of El-Gamal

We have no ability to prove that the cryptanalysis of any reasonable algorithm is difficult, because that would effectively require proving $P \neq NP$ (and possibly more!). But we can prove that decrypting one algorithm is "at least" as hard as decrypting another. We can prove that breaking an El-Gamal cipher is at least as hard as breaking a Diffie-Hellman key exchange.

In particular, suppose Alice and Bob are doing a Diffie-Hellman key exchange, and are overheard by Eve. But Eve has an *ElGamal oracle*: a machine that will take in an ElGamal public key and ciphertext, and reveal to her the corresponding plaintext. Thus this works if Eve has any efficient way to break an ElGamal cipher—whether it involves actually finding the private key or not.

So Eve overhears Alice's transmission of $A \equiv g^a \mod p$ and $B \equiv g^b \mod p$, and she wants to compute $g^{ab} \mod p$. We saw in section 5.6 that there's no known efficient algorithm for doing this; the best option we have is to compute a discrete logarithm.

But with her oracle, Eve chooses a random number c_2. She tells her oracle that the public key is A and the ciphertext is (B, c_2). By definition, the oracle returns to her the "plaintext":

$$m = (c_1^a)^{-1} c_2 \equiv (B^a)^{-1} \cdot c_2 \equiv (g^{ab})^{-1} \cdot c_2.$$

Eve can then invert this number m, and compute $m^{-1} \cdot c_2 \equiv g^{ab}$ the private key that Alice and Bob have exchanged.

This doesn't tell us that breaking ElGamal is hard, because we don't know for sure that breaking Diffie-Hellman is hard. But it does prove that ElGamal is *at least* as secure as Diffie-Hellman, because if we can break ElGamal then we can also break Diffie-Hellman.

Also notice that if it's possible to break ElGamal without computing an explicit discrete logarithm, it is also possible to break Diffie-Hellman without a discrete logarithm.

6.2.2 Complexity and Implementations

Looking at the algorithm for ElGamal, we see that encrypting a message requires two exponentiations, and thus with the fast exponentiation algorithm ElGamal is $O(\log_2(p))$. Decryption requires one exponentiation, and so is also $O(\log_2(p))$.

However, while the second exponentiation in the encryption step isn't important asymptotically, it does double the amount of computation necessary to encrypt a message—and the number of bits that need to be transmitted, since a single k bit number is encrypted to be a pair of k bit numbers.

In order to save on these extra computations and bit transmissions, we often use ElGamal in a hybrid setup. The "true" message is encrypted with a *symmetric* cryptosystem, which can provide the same security for less up-front computation. Then the *key* is encrypted with the ElGamal cryptosystem.

6.3 The RSA Cryptosystem

Though ElGamal is a useful cryptosystem, it is not the first public-private key cryptosystem that was invented or published.

The RSA algorithm was published by Rivest, Shamir, and Adleman in 1978. It was first discovered by Clifford Cocks in 1973 (in conjunction with James Ellis), but this fact was not declassified by the British government until 1997.

Algorithm 6.2 (RSA Cryptosystem)**.** Suppose Alice wants to send a message to Bob.

First Bob must create and publish a public key, and compute a private key for himself.

1. Bob chooses two primes p, q and computes $N = pq$. He also computes $M = (p-1)(q-1)$.
2. Bob chooses a number e such that $\gcd(e, M) = 1$.
3. Bob publishes the pair (N, e). This is his public key. Bob does not publish p or q or M.
4. Bob computes the inverse of e modulo M, and calls it d. Thus d solves $ed \equiv 1 \mod M$. Bob's private key is the pair (M, d).

Now Alice wishes to send Bob an integer m with $1 \leq m < N$.

1. Alice computes $c \equiv m^e \mod N$. She sends c to Bob.
2. Bob computes $c^d \mod N$ and receives Alice's message m.

Proposition 6.16. *The decryption step of Algorithm 6.2 works. That is, $c^d \equiv m \mod N$.*

Proof. Recall that $ed \equiv 1 \mod (p-1)(q-1) = \phi(N)$, and thus $a^{ed} \equiv a^1 \mod N$ by 6.10.

$$\begin{aligned} c^d &\equiv (m^e)^d \\ &\equiv m^{ed} \\ &\equiv m^1 \equiv m \mod N. \end{aligned}$$

□

Example 6.17. Bob chooses the primes $p = 73$ and $q = 89$. He computes $N = pq = 6497$. He also computes $M = (p-1)(q-1) = 72 \cdot 88 = 6336$, but does not share this with anyone

Bob chooses the exponent $e = 83$ and checks that $\gcd(83, 6336) = 1$. He then computes

$$d \equiv e^{-1} \equiv 83^{6335} \equiv 6107 \mod 6336.$$

Bob publishes the public key $(M, e) = (6497, 83)$. The pair $(M, d) = (6336, 6107)$ is his private key, which he keeps private.

Suppose Alice wishes to send Bob the message 300. Alice computes

$$c \equiv 300^{83} \equiv 4955 \mod 6497.$$

She sends Bob the message 4955.

Bob computes

$$c^d \equiv 4955^{6107} \equiv 300 \mod 6497$$

and recovers the message Alice wished to send.

Remark 6.18. These calculations are quite easy on computers with good algorithms, but are quite tedious to do by hand, especially since they tend to involve large numbers.

You can do all of this on Wolfram Alpha, including typing in "inverse of 83 mod 6336".

Example 6.19. Bob chooses the primes $p = 199$ and $q = 577$. He computes $N = pq = 114823$

6.3.1 Breaking RSA

In order to decrypt the ciphertext, Eve needs to solve a congruence of the form $x^e \equiv \mod N$. If Eve knows the values of p and q this is straightforward, and she can decrypt the message in exactly the same way that Bob can. As far as we know, there's no better way of breaking RSA than trying to factor N, but that doesn't mean there isn't a better way.

So how hard is factoring? It turns out to be quite difficult with current knowledge. The obvious thing to do is to just try dividing by a lot of numbers; this algorithm is about $O(N)$, which we should recall is exponential in $k = \log_2(N)$ and thus slow.

The next method is called Fermat factorization, and leverages the identity $a^2 - b^2 = (a + b)(a - b)$. Thus we can convert factoring problems into problems about writing N as a difference of squares. With some clever choices, this method works in something like $O(\sqrt[4]{N})$ time with basic optimizations—which is subexponential, but still much worse than polynomial time.

With substantial optimization, we get the Quadratic Sieve, which runs in time

$$O\left(e^{\sqrt{\log(n)\log\log(n)}}\right).$$

The best currently-known factorization algorithm for large numbers is the General Number Field Sieve, which runs in time

$$O\left(e^{\sqrt[3]{64/9}\ln(n)^{1/3}\ln(\ln(n))^{2/3}}\right).$$

This is only more efficient than the quadratic sieve for numbers larger than 10^{100}, but numbers used for cryptography are necessarily over that threshhold.

To get realistic security against modern computer-based attacks, we use moduli with about 1024 bits of entropy, which are 300-digit numbers when written in base 10. There are a series of standard moduli used for RSA encryption, with names like RSA-512 for the 512-bit RSA modulus.

Modern software can factor a 128-bit number on a desktop computer in less than 2 seconds, a 256-bit number in under two minutes, and a 320-bit number in under two hours.

In 2009, Benjamin Moody successfully factored RSA-512 in 73 days on a desktop computer. RSA-768 has been factored in 1500 CPU-years over two real-time years. No larger RSA number has been known to be factored. Both of these attacks used the general number field sieve.

In contrast, there's no known security advantage to using a large e instead of a small one. But it makes some people nervous, because we don't know there *isn't* an advantage.

References

[Singh(2015)] Vikram Singh. A practical key exchange for the internet using lattice cryptography. Cryptology ePrint Archive, Report 2015/138, 2015. http://eprint.iacr.org/2015/138.