Public Key Cryptography

SIAM Seminar - Top Algorithms in Computer Science

Michael Dougherty

Department of Mathematics
University of California, Santa Barbara

November 12, 2014

Introduction

Cryptography recently underwent a major transformation in the 1970s, drastically changing the course of a centuries-old field.

These algorithms are unlike many of the others that we study, but they are essential for modern communication.

I am by no means an expert in this field, so feel free to stop me and ask questions if I'm unclear.

Pre-Modern Cryptography

Virtually all cryptographic schemes before the 1970s were *symmetric*.

The methods of enciphering/deciphering grew more complex, but still required pre-arranged knowledge of a secret key.

Given the limits of computational power pre-20th-century, this wasn't really a problem.

World War II and the Rise of Cryptanalysis

The *Enigma* cryptosystem adapted by the Nazis drove the need for a more sophisticated method of code breaking.

Alan Turing led the development of computing at Bletchley Park with the aim of deciphering the *Enigma*. Their success underscored the future importance of computing in cryptography.

The introduction of computers as abstract objects (Universal Turing Machine) led to questions about the theoretical strength of ciphers instead of just their practical strength.

Shannon and the Theory of Information

Claude Shannon constructed a rigorous theory of information and how it is transferred:

How can you communicate a perfect message over an imperfect channel?

These ideas led others to the development of more advanced cryptographic techniques based on a similar question:

How can you communicate a secure message over an insecure channel?

Public-Key Methods

There are no secure channels in electronic communication!

How can two (or more) parties exchange public information which result in a shared secret?

The Discrete Logarithm Problem

Definition: Let $a, b \in \mathbb{Z}_n$. The *logarithm of b with base a* is the unique element $x \in \{0, 1, \ldots, n-1\}$ such that $b = a^x$.

Fact: Finding discrete logarithms is hard!

Diffie-Hellman Key Exchange

- Choose a (large) prime p and a generator $a \in \mathbb{Z}_p^*$, i.e. $gcd(a, p-1) = 1$.
- Alice chooses $x \in \{1, \ldots, p-2\}$ and sends a^x to Bob.
- Bob chooses $y \in \{1, \ldots, p-2\}$ and sends a^y to Alice.
- Alice computes $(a^y)^x = a^{xy}$.
- Bob computes $(a^x)^y = a^{xy}$.

Alice and Bob now share a common secret key which they can use for encryption in another algorithm.

ElGamal Key Generation

Alice can do the following to create her public and private keys:

- Choose a (large) prime p and a generator a of the multiplicative group $\mathbb{Z}_p^*$.
- Select a random $x \in \{1, \ldots, p-2\}$ and compute a^x mod p.
- Make p, a, and a^x public, keeping x private.

Anyone who intercepts the traffic will be unable to determine x due to the difficulty of the discrete logarithm problem.

ElGamal Encryption

Using Alice's public key, Bob can securely send a message $m \in \{0, \ldots, p-1\}$:

- Bob selects a random integer $k \in \{1, \ldots, p-2\}$.
- Bob computes $\gamma = a^k \pmod{p}$ and $\delta = m \cdot (a^x)^k \pmod{p}$.
- Bob sends the the ciphertext of γ and δ to Alice.
- Alice uses x to compute $\gamma^{-x} \pmod{p}$ and recovers m via $\gamma^{-x} \cdot \delta \equiv m \pmod{p}$.

Integer Factorization

Fact: Integer factorization is extremely difficult!

RSA Key Generation

Alice follows these steps to generate her public and private keys:

- Choose two large primes p and q.
- Compute $n = pq$ and $\phi = (p-1)(q-1)$.
- Choose a positive integer $e < \phi$ such that $gcd(e, \phi) = 1$.
- Find the unique integer $1 < d < \phi$ such that $ed \equiv 1 \pmod{\phi}$.
- Make n and e public, keep d private.

We say that e is the *encryption exponent* and d is the *decryption exponent*.

RSA Encryption

Once keys have been generated, Bob can use the public information to send an encrypted message $m \in \{0, 1, \ldots, n-1\}$ that only Alice can read:

- Bob obtains the public keys n and e.
- Bob then computes $c = m^e \pmod{n}$ and sends c to Alice.
- Alice computes $c^d = m^{ed} \equiv 1 \pmod{n}$.

But why does this work?

Proof of RSA

Since $ed \equiv 1 \pmod{\phi}$, there exists an integer k such that $ed = 1 + k\phi$. There are two cases:

1. If $gcd(m, p) = 1$, then by Fermat's little theorem, $m^{p-1} \equiv 1 \pmod{p}$, so $m^{k(p-1)(q-1)} \equiv 1$ and hence $m^{1+k\phi} \equiv m \pmod{p}$.
2. If $gcd(m, p) = p$, then the above holds trivially since $m \equiv 0 \pmod{p}$.

Either way, we get $m^{ed} \equiv m \pmod{p}$. Similarly, $m^{ed} \equiv m \pmod{q}$, and since p and q are distinct primes, we must have $m^{ed} \equiv m \pmod{n}$.

Elliptic Curve Cryptography

Why do we need to use a number-theoretic problem for the basis of our cryptosystem? There is an interesting structure among elliptic curves with similar difficulty benefits.

Potential Issues

This gives you an idea of the underlying theory, but how do we implement these algorithms?

- Choice of primes/generators/curves
- Padding/salting messages
- Computation optimizations
- How do you know how good a cipher is?
- How will you know that you've implemented an algorithm correctly?

Related Problems

These ideas raise a host of other questions and applications:

- SSH/SSL
- Digital signatures
- Non-number-theoretic problems?
- Side channel analysis
- Fully homomorphic encryption
- Quantum cryptography