Lecture 34: Cryptology

Aims:

- To explain some of the principles of cryptology;
- To show how modern cryptology exploits results from complexity theory.

34.1. Cryptology

- It is likely, when you reach Fourth Year, that you will have the opportunity to study a module in computer security. It includes a thorough discussion of cryptology. However, I want to give you a brief introduction to the topic now. I want to show you how modern cryptology exploits results from complexity theory.
- The importance of security in military, diplomatic, financial and industrial applications is obvious. Its importance has grown immeasurably with the growth of Internet applications, especially e-commerce applications.
- **Cryptology:** is the study of systems that keep secrets.

  Cryptography is about making such systems;

  Cryptoanalysis is about breaking such systems.
- Suppose Alice wants to send a message, M, to Bob.
- Suppose Eve (an eavesdropper) can intercept all messages.
- How can Alice and Bob keep M secret from Eve?

  Encryption: The message M, called the *plaintext*, is converted into C, called the *cyphertext*, before being sent.

  Decryption: Cyphertext C is converted back to plaintext M after it has been received.
- It must be easy for Alice to transform M to C.
- It must be easy for intended recipients, such as Bob, to transform C back to M.
- It must be difficult for unintended recipients, such as Eve, to transform C back to M.

34.2. Private-Key Encryption

- Traditionally, a secret (or private) *key*, shared only by Alice and Bob, is used to encrypt and decrypt.
- Alice encrypts M using the key:

$$C = encr(M, key)$$

- Bob decrypts C using the key:

$$M' = decr(C, key)$$

- We need it to be the case that

$$M' = M$$

- E.g. *Substitution Cypher:* the key is a permutation of the letters of the alphabet.

Alphabet	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
Key	T	H	E	O	R	Y	F	C	M	P	U	A	I	N	B	D	G	J

$M =$ "HELLO"; $C =$ "CRAAB"

- This cypher is easily broken by frequency analysis.
- With layer upon layer of additional sophistication, less easily broken cyphers can be devised. We won't be looking at these ideas.
- But, the main problem with all private-key encryption systems, no matter how sophisticated they may be, is *key transfer*. How do we distribute the private key? We cannot distribute it along the same, unsafe channels as the encrypted message. So we must resort to more expensive methods, e.g. trusted couriers. Of course, this is impractical in many applications, such as e-commerce.

34.3. Public-Key Encryption

- In public-key encryption, all users have two keys, a public key and a private key.
- Public keys are used for encryption. They must be published in a trusted directory. This raises issues of its own. We must have faith in the integrity of the directory. We don't want forgeries, for example, where an interloper publishes his/her own public key while claiming to be me. Any messages you send to me using the forged public key would be decipherable by the interloper.
- Private keys are used for decryption; they must be kept secret.
- Let's try to understand how it work by using an analogy.
- Alice wants to send a message to Bob. Bob has both a padlock (his public key) and the key to the padlock (his private key).
- He puts the open padlock in a public place, but he keeps the key to himself.
- Alice puts her message in a box, gets Bob's padlock and clicks it shut on the box, and sends the box to Bob.
- No-one other than Bob has the key, so the message is safe.
- When Colin wants to join in, he just buys himself a padlock and key and makes the open padlock public. Now anyone can send him secure messages too.
- In computer terms, Bob has a public key, $pubKey$, and a private key, $privKey$, instead of padlocks and keys.
- There will obviously be some mathematical relationship between the two keys.

- Alice gets Bob's public key from the trusted directory. She encrypts M using Bob's public key:
$$C = encr(M, pubKey)$$
- Bob decrypts C using his private key:
$$M' = decr(C, privKey)$$
- We need it to be the case that
  1. $M' = M$
  2. Ideally, it should be reasonably computationally cheap to devise the two keys (since this needs to be done for every user, and needs to be re-done periodically).
  3. Ideally, *encr* should be computationally cheap.
  4. Ideally, *decr* should be computationally cheap.
  5. Ideally, it should be impossible to compute M from C without knowing the private key. If not impossible, it should be provably computationally intractable. If not provably intractable, there should be no known polynomial-time algorithm.
  6. Ideally it should be impossible to compute the private key from the public key, even if you know the mathematical relationship between the two. If not impossible, it should be provably computationally intractable. If not provably intractable, there should be no known polynomial-time algorithm.
- Unfortunately, all the schemes so far devised rest on conjectured, not proven, intractability.
- So to be a cryptographer or a cryptanalyst, you have to study the Theory of Computation!
- Some features of public-key encryption:

- There is no need for a secure channel.
- There is no need for key transfer of a shared key.
- The mathematical relationship between the public key and the private key can be publicised without compromising the private key.
- Even Alice can't decrypt her own encrypted message.
- Alice and Bob don't even have to know each other.

34.4. RSA Public-Key Encryption

- RSA is one example of a public-key encryption scheme.
- RSA is based on the difficulty of factoring large numbers.
- An integer p is *prime* if $p \geq 2$ and its only divisors are 1 and p.
- *Fundamental Theorem of Arithmetic:* Let n be an integer, $n > 1$. Then there is a unique set of prime numbers $p_1, \ldots, p_k$ and positive integer exponents $q_1, \ldots, q_k$ such that
$$n = p_1^{q_1} \times \ldots p_k^{q_k}$$
- E.g. $26 = 2^1 \times 13^1$
- E.g. $1200 = 2^4 \times 3^1 \times 5^2$
- Given n, finding this set of prime numbers is known as *factoring*.
- Two integers, m and n, are *relatively prime* if their greatest common divisor is 1. (The greatest common divisor is the largest integer that divides both m and n.)
- E.g. 4 and 9 are relatively prime.
- Euler's totient function of a positive integer, n, written $\phi(n)$, is defined as the number of positive integers less than or equal to n that are relatively prime to n.

34.4.1. Devising the keys

- Select two large prime numbers, p and q.
- Let $n = pq$.

- Compute $\phi(n) =_{\text{def}} (p-1) \times (q-1)$.
- Select public key *pubKey* and private key *privKey* as follows:
  - Choose a number *pubKey* so that *pubKey* and $\phi(n)$ are relatively prime. (An obvious choice for *pubKey* is another large prime with a few more digits than p and q have.)
  - Compute *privKey* from the following:

$$pubKey \times privKey = 1 \bmod \phi(n)$$

- Publish n and *pubKey* in a trusted directory.
- Keep *privKey* secret. p and q are also kept secret, but they are no longer needed.

34.4.2. The cost of devising the keys

- Devising the keys is not the most time-critical part of the process: we do it only for new users and when we change our keys, which is only every so often. It's more important that encryption and decryption be efficient. Nevertheless, we don't want it to take too long to devise the keys.
- n must be a large number so that it is hard to factor it back into p and q. This, in turn, requires that p and q be large numbers.
- So we need an efficient algorithm for finding large prime numbers.
- Primes are not too rare.
- So we simply generate a random odd integer, i, of an appropriate size and test it. We keep doing this until a prime is found.

- How long will it take to test i?
- The naïve approach is to divide i by $2, 3, \ldots \sqrt{i}$. (Of course, you don't have to divide it by any even numbers apart from 2.) But this is exponential in the length in bits of i.
- The alternative that is used in practice is to use a fast, probabilistic algorithm. I won't give the algorithm, but it is very fast. But, being a probabilistic algorithm it sometimes gives the wrong result. Specifically, it sometimes says that an integer is prime when it is not. However, the probability that it does this is extremely small. For the size of numbers we are testing, the probability of a wrong answer is smaller than the probability of winning the Lotto and being struck by lightning on the same day.
- Remarkably, in 2002 a polynomial-time algorithm was discovered, although this is not yet seeing widespread use. So the status of this problem changed. Before 2002, we had only exponential algorithms, but we had no proof that the problem was intractable. (In fact, the problem was in **NP** and we didn't know whether it was in **P** and we didn't know whether it was **NP**-complete.) After 2002, we now know that the problem is tractable (and, since it is a decision problem, it is in **P**).

34.4.3. Encryption

- Alice wants to send Bob a message.
  - The message must be expressed as an integer, M. E.g. we can simply take the ASCII for M. If $M > n$, then Alice must split it into separate messages, each less than or equal to n. For simplicity, we'll assume $M \leq n$.
  - Alice gets Bob's n and $pubKey$ from the trusted directory.
  - She encrypts: $C = M^{pubKey} \bmod n$
  - She sends C to Bob.

34.4.4. Decryption

- Bob wants to decrypt C.
  - He gets his private key $privKey$
  - He decrypts: $M' = C^{privKey} \bmod n$
- It can be proven that $M' = M$.

34.4.5. The cost of encryption and decryption

- Both encryption and decryption consist of exponentiation and division to compute $\bmod n$. And we're applying these operations to very large numbers.
- There are algorithms that compute x^y in $O(\log y)$ multiplications/divisions.
- We can even incorporate the $\bmod n$ into the exponentiation algorithms so that the multiplications/divisions carried out by the exponentiation algorithm apply to numbers that are always less than n.
- So it turns out that both encryption and decryption require $O(\log n)$ multiplications/divisions on numbers less than n.

34.4.6. Can the system be cracked?

- Eve wants to eavesdrop.
  - She intercepts the encrypted message, C.
  - She can also obtain Bob's n and $pubKey$.

  - It is believed but not proven that the only way to crack RSA is to figure out *privKey*. In other words, we believe there is no other way of going from C to M. There is a risk that someone will have a mathematical insight that enables them to compute M from C without needing *privKey*.
  - It is believed but not proven that the only way to obtain *privKey* is to factor n into p and q. Again there is a risk that someone will have a mathematical insight that enables them to obtain *privKey* from n and *pubKey* without factoring.
  - It is believed but not proven that factoring is intractable. In other words, we have exponential-time algorithms, we know of no polynomial-time algorithm, but we have not been able to prove that the problem is intractable (we have not proved that a polynomial-time algorithm cannot exist.) There is a risk that the problem is, in fact, tractable, and that someone will devise a polynomial-time algorithm. If that ever happens, then RSA is done for.

    Of course, on the other hand, if someone ever proves that factoring is intractable, then we will know that RSA is very secure.
  - I don't know the complexity of the current fastest algorithm. But there is one that takes $O(e^{\log n \log\log n^{\frac{1}{2}}})$ steps.
  - If n has 1000 digits, such algorithms would not terminate in our lifetime. And even as hardware improves, it makes very little dent into an exponential time algorithm, and we can keep ourselves safe by choosing slightly larger values for p and q and, of course, by periodically changing our keys.

- There are some subtleties that I have not mentioned. They don't really matter to your understanding. But for those of you who are very interested here they are.
  - For every public key, at least 9 messages are unconcealable. In others words, there are at least 9 messages, where $M = M^{pubKey} \bmod n$ (i.e. the cyphertext comes out the same as the original message). Of course, the software can easily spot this and tell the user to rephrase his/her message.

– Also for some choices of keys, no messages are concealable, i.e. the cyphertext is always the same as the plaintext. But it is possible to avoid choosing these keys.

34.5. Other Security Issues

- Security is a broad topic, not confined to encryption and decryption.
- For example, not only must we prevent Eve from deciphering messages we must also find ways of preventing her from changing messages, replacing them, delaying them, blocking them, repeating them, rerouting them and reordering them.
- Ideas from computational complexity theory arise not just in encryption/decryption but also in digital signatures, digital fingerprints, digital certificates, and secure electronic transaction systems used for secure credit-card payments.
- You can study some of these and other issues in the security module in Fourth Year.

34.5.0.1. Acknowledgements:

I took the analogy involving padlocks and keys from [Sin99]. The rest of the material is mostly based on the treatment given in [GT02].

Clip Art (of head with bomb) licensed from the Clip Art Gallery on DiscoverySchool.com.

References

[GT02] M. T. Goodrich and R. Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples*. Wiley, 2002.

[Sin99] S. Singh. *The Code Book*. Fourth Estate, 1999.