Contents

6 Modern Topics in Cryptography

In this chapter, we discuss a few topics relevant to modern cryptography as currently implemented. We will discuss internet security protocols (examples including PGP, SSL, and TLS) as well as trusted authorities and certificates. We will also give a vague outline of the ideas behind quantum computing and explain how, if a sufficiently large quantum computer could be built, it could be used to break many modern protocols.

6.1 Authentication, Trust, and Certificates

- When performing digital interactions (or even non-digital interactions), Alice and Bob want to ensure that they are actually talking to each other and not some third party.
    - We have developed the idea of a digital signature algorithm, using which Alice can verify that each message in their conversation was signed by Bob, and vice versa.
    - When designing signature algorithms, we generally made statements along the lines of "Bob publishes his public key and Alice uses it to verify his signature is correct".
    - However, there is a difficulty that needs to be addressed here: how does Alice know that the public signature key posted on "Bob's" website is actually from Bob?
    - After all, Eve might have pretended to be Bob and directed Alice to a webpage that claims to contain Bob's public key, but actually was created by Eve. Or perhaps Mallory hacked Bob's webpage to display the wrong public key, and Bob has not noticed.
- One possible way to deal with this problem is if Alice trusts a third party, Trent, who is willing to vouch for Bob's identity.
    - Alice then does not need to believe Bob directly: she needs only to believe that Trent is telling the truth when he says that Bob is who he claims to be (or in the case of a webpage, that Trent is telling the truth that it really is Bob's webpage and not Eve's).
    - Of course, if Alice lives in Argentina and Bob lives in Belarus, there may not be such a person who knows both Alice and Bob well enough for the procedure to work. (Alice would need to trust that she is really speaking to Trent, and Trent would need to trust that he is really speaking to Bob.)
    - However, we can easily extend the procedure: if there is some chain of people starting with Alice and ending with Bob such that each person trusts the next person in the chain, then Alice and Bob can establish a communication channel through that trusted chain.

- One possibility is that Alice and Bob could rely on a "web of trust": Alice trusts a small group of people, each of whom in turn trusts a few others, and so on and so forth. With enough trusted intermediaries, the principle is that any two people in the network can establish an initial authentication to one another, after which they can employ digital signing to secure subsequent communications.
- Another possibility is that Alice and Bob could rely on a "hierarchal" model: Alice and Bob have a chain of people leading to a centralized certificate authority that will eventually vouch for their identities to one another.

6.1.1 Public Key Infrastructures

- If Alice wants to send Bob a message encrypted with Bob's public key, Alice wants some way of assuring herself that the public key she is using really does belong to Bob.
- One way of ensuring this is to use a public key infrastructure (often abbreviated "PKI"), which is a scheme whereby all of the parties can establish their identities to one another using digital certificates.
    - Each certificate contains some amount of information whose contents are signed by a central authority using a digital signature algorithm that is hard to forge.
    - Certificates can indicate various things. We will primarily discuss identity certificates, which contain information about a party's identity, such as a name or address, as well as their public keys. Another type of certificate is an access certificate, which indicate the level of information a party is allowed to access (i.e., whether they are allowed to view sensitive material).
- The infrastructure, in its simplest form, consists of the following components:
    - A certificate authority (CA) that stores, issues, and signs certificates.
    - A registration authority (RA) that verifies the identities of new parties that want certificates.
    - A public directory, containing a public index of certificates.
- So suppose Alice wants to send Bob a message using Bob's public key.
    - First, Bob needs to register himself with the registration authority, Reg. Reg will verify his identity and hand him a digitally-signed document containing his identity information along with Reg's signature.
    - Next, Bob takes his identity document and gives it to the certification authority, Cert. Cert verifies that Bob's identity document has been correctly signed by Reg and creates an identity certificate for Bob containing Bob's identity information along with Bob's public keys and Reg's signature.
    - Cert then puts Bob's identity certificate into the public index.
    - Now, if Alice wants to send Bob a message, she looks in the directory to find Bob's identity certificate in the directory and decide whether she trusts Cert and Reg. If she does, she retrieves Bob's certificate and extracts Bob's public keys from it, which she can then use to send him a message.
    - Note of course that Alice does not interact with Bob when retrieving his public key, and it is not even necessary for her to trust him directly.
- In actual real-world public key infrastructures, there are many certificate authorities and registration authorities, since the scale of the record-keeping would be unreasonably large for any one individual group to manage.
    - In such a case, the certificate authorities would certify that the others are allowed to issue certificates, and each certificate authority would accept registrations from any of the registration authorities.
    - Of course, it could be the case that Cert thinks that Dave (another CA) is a little bit too lax in their procedures, so Cert only certifies Dave to issue certificates to individual people, not other CAs. On the other hand, Dave thinks Cert is very scrupulous, and so Dave trusts any CA that is also certified by Cert.
    - Likewise, Cert and Dave may only trust particular registration authorities: Cert trusts Reg and Sam, while Dave trusts Reg and Taylor.
    - One can quickly imagine how tangled such trust relationships can get with dozens or hundreds of CAs and RAs, each of which has various kinds of trust in the others for particular certifications.

6.1.2 X.509 Certificates

- A commonly-used digital certificate is the X.509 certificate, which is used in the the SSL and TLS protocols.
  - The protocol was initially deployed in 1988 and involves a hierarchy of certificate authorities: the top-level authorities certify particular authorities that are one tier lower, and those authorities in turn certify lower ones, and so forth.
  - A certificate chain consists of a sequence of certificates, each signed by an authority higher on the list, terminating in a top-level authority.
- If Alice wants to interact with Bob, she requests a copy of his certificate chain and checks the validity of each certificate until she reaches an authority she trusts.
  - Suppose that Bob's certificate is signed by Cert, and Cert's certificate is signed by Verisign (currently one of the largest top-level certificate authorities).
  - Alice first verifies that Bob's certificate is valid. She sees that it is signed by Cert, but she does not necessarily trust Cert. She then checks Cert's certificate and sees that it is properly signed by Verisign, whom Alice does trust.
  - Alice accepts Bob's certificate, since it is part of a trusted chain, and can now use it to send him messages.
- An X.509 certificate contains the following information:
  - The version number of X.509 being used (to allow forward and backward compatibility if the protocol is subsequently changed).
  - A unique serial number, used to identify the certificate.
  - Identity information for the party to whom the certificate belongs (e.g., Bob).
  - The identity of the certificate authority, their signature attesting the validity of the certificate, and the algorithm that can be used to verify the signature.
  - The date range during which the certificate is valid.
  - The public keys attached to the party to whom the certificate belongs, as well as their purposes (e.g., public-key encryption or digital signatures) and underlying algorithms (e.g., RSA, ElGamal, etc.).
  - A hash of the certificate along with the algorithm used, to prevent a forgery of the certificate once the certificate is issued.
- The most common use of certificates is to create secured internet connections (HTTPS connections), which is done in the following way:
  - Suppose that Alice wants to connect to Bob's website with her browser using an HTTPS connection.
  - Bob has, ahead of time, obtained a valid X.509 certificate for his web server from a certificate authority.
  - Alice connects to Bob's server and the server sends Alice a copy of Bob's certificate.
  - Alice (or rather, her browser) then checks whether the certificate is valid: she checks whether the certifying authority of Bob's certificate is one that she trusts, she makes sure the identity of the party in the certificate matches the identity of the server, she checks that the certificate has not expired, she checks that the certificate has been correctly signed, and she checks whether the certificate has been revoked after it was issued.
  - If all of these things are valid, then Alice accepts the certificate.
  - Alice can then use the public key in the certificate to send an encrypted message back to Bob's server: in particular, she can generate a symmetric key for use in encrypting all subsequent communications and send it to Bob.
  - Now Alice has established an encrypted communication channel with Bob, and she can feel secure that she is actually interacting with the owner of Bob's certificate, which (provided she trusts the certificate authority) is Bob.

- We will also remark that it is possible for certificate authorities to revoke certificates after they have been issued. (The only built-in way for a certificate to become invalid once it has been created is if its expiration date passes.) However, the mechanism used by X.509 for revoking certificates is both inefficient and insecure.
    - Some instances in which a certificate should be revoked are if the certificate authority improperly issues the certificate (either by accident or through a malicious act) or the certificate's public key is compromised or lost.
    - Revoked certificates are put on a "certificate revocation list", which is published as part of a public directory available to anyone who requests it.
    - If Alice wants to ensure that Bob's certificate has not been revoked, she can then ask Bob's certificate authority whether his certificate is still valid.
    - This is quite inefficient, since it negates the entire advantage of not requiring all authentication requests to be processed by one of the central authorities.
    - Furthermore, if a malicious attacker is able to hijack Alice's web traffic, then it would be fairly easy to disable Alice's ability to check certificate revocation lists.
- In actual practice there are a few additional practical weaknesses in the system, since the initial difficulty of establishing a trusted authority still exists.
    - Modern browsers are generally prepackaged with a list of trusted "root certificates", which are automatically accepted by the browser. This is done primarily for convenience, as most users do not need or want to deal with the intricacies of how secured authentication actually works.
    - Such root certificates are generally self-signed (i.e., signed by the certificate authority itself, rather than another party) and serve as a "trust anchor": any certificate that is signed by one of these authorities will be accepted as valid.
    - If the browser encounters a new certificate chain, it follows the chain until it reaches one of its root certificates. If all of the certificates in the chain are correctly signed, it accepts the new chain.
    - However, it is very unlikely that any user will actually be familiar with all of the certificate authorities that their software implicitly trusts, meaning that the browser designers are essentially in charge of deciding which certificate authorities are trusted.
- On the other hand, large certificate authorities may not be "trustworthy" in the traditional sense, but since they charge money to create certificates, they have a strong economic incentive not to generate certificates that do not have sufficient identity verification.

6.1.3 PGP and SSL/TLS

- There are other mechanisms by which we can establish a trusted chain between Alice and Bob.
- Rather than relying on centralized certificate authorities, another option would be to rely on a decentralized "web of trust": each party has a certificate, and the trust in any party's certificate is certified by other users (who indicate their trust by signing the certificate).
    - Suppose that Alice wants to decide whether she trusts Bob's certificate. She does not know Bob, but she does trust Carol, who says Bob is completely trustworthy. Alice decides to accept Bob's certificate.
    - Alternatively, suppose that Alice trusts Carol and Dave, each of whom say Bob is somewhat trustworthy. She again decides that Bob has enough collective trust from people she trusts that she should accept Bob's certificate.
    - Alice then adds Bob to her list of "trusted parties" and adds her signature to his certificate, so that anyone who trusts her can then trust Bob.
    - As more users join the network and sign one another's certificates, the probability that any two people will share enough signatures in common to trust one another grows.
- This web-of-trust model was first promulgated by the Pretty Good Privacy (PGP) software, which was created by Phil Zimmermann in 1991 and is commonly used for encrypting email.

- The name is a reference to the name of a grocery store, Ralph's Pretty Good Grocery, in Garrison Keillor's fictional town Lake Wobegon.
- The early history of PGP is rather interesting: it, along with its source code, were freely distributed by its creator inside the United States (which in 1991 was a more substantial undertaking), and it provided quite strong encryption capabilities.
- It rapidly spread outside the United States (it was after all distributed on the internet) and acquired a substantial international userbase, a number of whom were dissidents in authoritarian states who used the software to prevent the government from intercepting their communications.
- The international distribution of the software in turn caused Zimmermann to be formally investigated by the FBI for trafficking in "munitions" to foreign countries: at the time, exporting cryptosystems using a key strength of 40 bits or larger without a license was a violation of US export regulations.
- We will remark that the lowest encryption standard supported by PGP, in 1991, was 128-bit symmetric encryption (which is considered to be reasonably secure even in 2016). In contrast, 40-bit encryption was weak even by 1991's computing standards.
- Although the investigation of PGP did not land in court, the restrictions on exporting cryptography eventually did in 1995 and 1996. The US circuit courts eventually held that the publication of software source code is protected by the First Amendment, and the government eventually loosened the restrictions substantially.

- PGP, as with most of the other authentication and encryption protocols, operates using a combination of public-key encryption, symmetric encryption, and hashing and signing protocols.
  - As with the systems that use certificates in a hierarchal system, if Alice wants to communicate securely with Bob, Alice first requests Bob's identity certificate.
  - If his certificate is signed by enough people that Alice trusts, and the signatures are valid, she accepts the certificate and extracts Bob's public key from it.
  - Alice then generates a random symmetric-encryption key and sends it to Bob using the public key she obtained from his certificate. Alice and Bob can now communicate securely using Alice's symmetric key.

- The Secure Sockets Layer (SSL) protocol, also equivalently known as the Transport Layer Security (TLS) protocol, operate on the same principle: a public key from a certificate is used to transmit a "session key" that will be used for symmetric encryption of subsequent communications.

- The details, however, are what make the protocols secure (or not).
  - The various versions of SSL/TLS and PGP support many different authentication, key exchange, and encryption protocols, including RSA, Diffie-Hellman, elliptic-curve Diffie-Hellman, and various digital signature algorithms.
  - The wide range of supported algorithms allows for better compatibility between different software that needs to use the protocol (e.g., if some programs using the protocol are older and have not been updated to use the newest algorithms or do not support the largest key sizes).
  - As part of a protocol, Alice and Bob must agree which algorithms and key sizes to use: for example, Bob may want to use 4096-bit RSA, but if Alice's software can only support 2048-bit RSA, Bob may have to revert to the lower key size.
  - There is a potential downside to this flexibility: various exploits applying to older versions of SSL/TLS were discovered that allowed for a man-in-the-middle attack to force two parties to downgrade to weaker algorithms than either would normally try to use.
  - Also included as part of these protocols are data integrity algorithms, which (roughly speaking) use hash functions like MD5 and SHA-1 to ensure that messages are not altered from their original form.
  - Our interest is primarily in the cryptography, rather than the implementation details, but we will note that there are a number of technical issues that require some care to deal with in order to prevent insecurities in the protocols.

6.2 Quantum Computing and Cryptography

- In this section we will give a superficial outline of quantum computing and its applications to cryptography.
- Very roughly speaking, a general principle of quantum mechanics is that any system exists as a superposition of all of its possible states (weighted with some probabilities) rather than a particular individual state[1].
    - By modeling the state of a system mathematically, we can then compute the probabilities of obtaining particular outcomes when we make an observation of the system.
    - To illustrate the contrast with a particular example: in the classical ("wrong") model, an electron is a discrete particle that exists in a particular location in space at particular times.
    - In the quantum-mechanical model, an electron is essentially a "probability cloud": it is not in any one particular place unless it is observed, and, if we make an observation, we are more likely to observe in in some places than others.
    - The "probability cloud" interpretation allows one to explain the fact that light behaves both as a wave and as a particle (e.g., as in the famous double-slit experiment): until an observation is made, the system exists in an appropriate superposition of states that is consistent with wave-like behavior, and after an observation is made, the behavior is forced to be particle-like.
    - The entire system can be thought of as a "wave function" (essentially, a function that keeps track of all the possible states of the system and their associated probabilities), which can change over time as components of the system interact with one another.
    - Under one of the standard interpretations of quantum mechanics, making an observation of the system will "collapse" the wave function into one of the possible states that are consistent with the result of the observation.
    - Once we make a measurement, then the superposition collapses and the other information will disappear (meaning that we cannot make two measurements of the system, nor can we "copy" the system and make separate measurements).
- The philosophy of quantum computing is essentially that, unlike bits on a classical computer that only come in two flavors (0 and 1), a quantum bit (or qubit) has many more possible states.
    - More precisely, a qubit can be thought of as a superposition of a 0 bit and a 1 bit with some probability attached to each possibility.
    - Using bra-ket notation, one would write a qubit as $a\,|0\rangle + b\,|1\rangle$, where the normalizations are chosen so that $|a|^2 + |b|^2 = 1$. The interpretation of the qubit $a\,|0\rangle + b\,|1\rangle$ is that, if we observe the qubit, the probability that it is 0 is $|a|^2$ and the probability that it is 1 is $|b|^2$.
    - The rough idea of quantum computing is that, by performing appropriate operations on our inputs, stored as qubits, we can change the state of the system so that it has a large probability of returning the result we are interested in.
    - The advantage over a classical computer is that a quantum computer is able to perform a potentially unbounded number of classical computations "simultaneously" and in parallel, and then return the single result that is the desired one.
- We will now briefly outline how Shor's algorithm for fast integer factorization works on a quantum computer.
- The idea is essentially to try to compute the order of an element a modulo $N = pq$: for a particular a such as $a = 2$, we look for an exponent k for which the sequence $1, a, a^2, a^3, \ldots$ has period k.
    - The smallest such k is likely to be a fairly large divisor of $\varphi(N) = (p-1)(q-1)$, since if a is a random residue class then a has a decent chance of being a primitive root modulo p and modulo q.
    - If u is the closest integer to N/k, then $\varphi(N)$ has a good chance of being equal to ku. We can then compute the divisors p and q by solving the system $pq = N$, $(p-1)(q-1) = \varphi(N)$, which is equivalent to a quadratic equation in p.

[1] Essentially every non-technical explanation of quantum mechanics requires a footnote saying "this description is not completely accurate". Ergo: this description is not completely accurate.

- We also note that there are ways to improve this procedure using continued fractions (which have the advantage of not needing us to find the smallest possible k).

- So we are reduced to needing to find the period of a particular sequence of integers modulo N. A good tool for this is the discrete Fourier transform.
  - Given a sequence a_0, a_1, a_2, ..., a_l of (positive) real numbers, we define the Fourier transform to be the function $F(x) = \frac{1}{\sqrt{l}} \sum_{t=0}^{k-1} e^{(2\pi i t x)/l} a_t$ for integers $0 \leq x \leq l$.
  - If the sequence is periodic with period k dividing l, then $F(x)$ will be zero whenever x does not divide l/k (the "frequency" of the sequence), and $F(x)$ will be nonzero at multiples of l/k.
  - These statements are a straightforward computation using the roots of unity $e^{(2\pi i t x)/l}$: essentially the idea is that the sums of the roots of unity all cancel one another when x does not divide l/k, but all have large sum when x does divide l/k. (We omit further details.)
  - As an example, if we have the sequence $1, 2, 3, 1, 2, 3$ modulo $N = 6$, we can compute $F(1) = F(3) = F(5) = 0$, while $F(2) = \frac{4 - 2i\sqrt{3}}{\sqrt{6}}$, $F(4) = \frac{-4 + 2i\sqrt{3}}{\sqrt{6}}$, and $F(6) = \frac{14}{\sqrt{6}}$.
  - In general if the period is a divisor of l, we can use the peaks of the Fourier transform to find it. If the period is not an exact divisor of l, then we will not get complete cancellation : however, the Fourier transform will still have large peaks near the multiples of l/k, the spacing of which we can use to get a reasonable close estimate of l/k.

- Shor's algorithm operates by computing the discrete Fourier transform of the sequence of residues $1, a, a^2, \ldots$ modulo N and searching for the distance between two peaks.
  - Explicitly, let $N = pq$ be the integer we wish to factor. Choose m so that $N^2 \leq 2^m \leq 2N^2$ and also choose a random residue a modulo N.
  - We begin by constructing a superposition of m qubits each of which is equally distributed between 0 and 1, which corresponds to the state $\frac{1}{\sqrt{2^m}} [|0000\rangle + |0001\rangle + |0010\rangle + \cdots + |1111\rangle]$ (where we have written all of the values of the bits together in order), and we view the sequence of m bits as an integer written in base 2.
  - We then compute the function $f(x) = a^x \bmod N$ on this system, which yields the state $\frac{1}{\sqrt{2^m}} \left[|a^0\rangle + |a^1\rangle + |a^2\rangle + \cdots + |a^{2^m - 1}\rangle\right]$.
  - At this stage, we have not actually done anything useful yet: if we measure the state, then we will obtain a single value $a^j \pmod N$ for some random $0 \leq j \leq 2^m - 1$, and we would lose all of the other information. (We could just as well have done this on a classical computer.)
  - The idea now is to compute the "quantum Fourier transform" of our superposition: given a state $|x\rangle$, it computes the value $QF(x) = F(x) = \frac{1}{\sqrt{2^m}} \sum_{t=0}^{k-1} e^{(2\pi i t x)/2^m} |t\rangle$, and is computed on a superposition of states by linearity.
  - We would then like to search for peaks of the quantum Fourier transform, which would correspond to multiples of the frequency $2^m/k$ of the element a. (Since 2^m was chosen so that $N^2 < 2^m$, there will be a reasonably large number of peaks.)
  - This is where the magic of the quantum computation comes in: when we make an observation, the superposition collapses and will return the result x with probability equal to the square of the absolute value of the coefficient of $|x\rangle$ in the quantum Fourier transform. Thus, we are extremely likely to observe a value that is in one of the peaks, which will be very close to a multiple of the frequency $2^m/k$.
  - Suppose (for sake of argument) that we landed in the peak near $2^m/k$: we would then have a small range of values of k to test, each of which would give a small number of possible values of $\varphi(N)$ and hence of p. We can then quickly check whether any of them give an integral value for p, and if so we have found the factorization.

- If we land in one of the other peaks, then by using a continued fraction computation, we likewise generate a short list of candidate values for $\varphi(N)$, one of which will be correct with reasonably large probability.

- Roughly speaking, the amount of data needed to perform Shor's algorithm is on the order of $m \approx 2\log_2 N$, and it can be shown that computing the quantum Fourier transform can be done in polynomial time.
  - In fact, the slowest portion of Shor's algorithm is the modular exponentiation.
  - More specifically, the running time of Shor's algorithm is known to be bounded above by a constant multiple of $(\log N)^2(\log\log N)(\log\log\log N)$.
  - Thus, if a suitable quantum computer could actually be built, Shor's algorithm would give a polynomial-time factoring algorithm that succeeds with high probability on any given attempt.
  - Such an algorithm would vastly outpace all of the others we have developed, and would essentially break all of the algorithms that rely on the slowness of integer factorization such as RSA.
  - The algorithm can also be adapted to compute discrete logarithms (which are quite obviously a direct question about evaluating the period of a sequence), and so would also break Diffie-Hellman and ElGamal encryption.
  - Likewise, the elliptic curve cryptosystems are equally susceptible (though some modification is required to compute point addition using a quantum computer), since they too boil down to an order computation.
  - However, we will say that it appears to be very difficult to build a quantum computer: the largest integer (as of 2016) factored using Shor's algorithm was 21.

Well, you're at the end of my handout. Hope it was helpful.