Contents

4 Digital Secrecy and Security

In this chapter, we discuss a variety of topics related to digital secrecy and security: We will first discuss hashing and digital signatures, giving several different procedures for digitally signing a message using procedures modeled on public-key cryptosystems. Next, we will discuss authentication and identity-verification protocols and the idea of a "zero-knowledge proof", allowing for a proof of identity that reveals no additional information to any parties. Then we will discuss "secret sharing" schemes for securely distributing a secret among several parties, and then some applications of all of the these ideas to secured multiparty computation. As particular examples of all of these ideas, we will then discuss digital voting and digital cash/currency systems and some implementations of these systems.

Our treatment of these topics will not be especially deep in any respect: our goal is to give an outline of some of the fundamental mathematical ideas rather than designing actual implementations. Many of the basic versions of the protocols we discuss, like with the basic implementations of public-key cryptosystems, are not sufficiently secure for practical use without some additional kinds of modifications such as message padding.

4.1 Digital Signatures and Hashing

- When Bob receives a message from Alice, he would usually like to have some kind of way of verifying that the message was actually sent by Alice, and not by Eve pretending to be Alice.
    - Classically, such a verification was done by having Alice signing her physical document before sending it to Bob by using a wax cast or some equivalent method. The underlying idea is that it would not be easy for Eve to forge Alice's insignia, nor would it be possible for Eve to remove Alice's insignia from another message and attach it to a fraudulent one.
    - Of course, on a digital letter, it is much less obvious how to create an unalterable signature that is bound to a document. Simply writing "I hereby sign this document with my secret password, emblazoned spanakopita" at the end (or anything of that sort) would obviously not be effective, since anyone can easily copy those words onto a new document.
- A digital signature must be created in such a way that binds it both to its creator (so that Bob knows Alice and not Eve was the signer and the sender) and to its associated message in a way that cannot easily be altered (so that Bob knows Eve didn't change the message).
- There are three basic goals when designing a digital signature algorithm: authentication, integrity, and non-repudiation.
    - Authentication refers to Bob's confidence that the message was really sent by Alice.
    - Integrity refers to Bob's confidence that the message was not altered since Alice sent it.
    - Non-repudiation refers to Bob's confidence that Alice cannot claim that she did not actually sign the message.
- Our goal when designing a digital signature algorithm is not to keep the message from being deciphered, but rather to prevent the signature from being easily decoupled from Alice's identity or from Alice's original message. Ultimately, however, these ideas are similar enough that we can adapt public-key cryptosystems to create digital signature algorithms.

4.1.1 RSA and ElGamal Signatures

- Our first digital signature algorithm is based off of the RSA cryptosystem.
    - Alice first creates an RSA public key (N, e), where $N = pq$ is the product of two large primes and e is relatively prime to $\varphi(N)$, and publishes it.
    - Alice also computes the associated decryption exponent $d \equiv e^{-1} \pmod{\varphi(N)}$, and keeps it secret.
    - If Alice now wants to sign a message m, she computes $r = m^d \pmod{N}$: her signature pair is then (m, y).
    - If Bob wants to verify that Alice really signed the message m, he simply computes $y^e \pmod{N}$ and compares it to the message m. If they are equal, then Bob accepts the signature, and if they are not he rejects it.
- Suppose now that Eve has intercepted a message pair (m, y) that Alice has signed and wants to forge Alice's signature on a new message w.
    - Obviously, Eve cannot simply use the signature pair (w, y), since Bob will compute $y^e \equiv m \pmod{N}$ and reject the signature as invalid.
    - In order to find a valid signature z for her message w, she needs to solve the congruence $z^e \equiv w \pmod{N}$. This is equivalent to decrypting the ciphertext w under Alice's public key, which is assumed to be difficult to do.
    - Another possibility is for Eve to try to choose the signature z first, and then compute the "message" $w \equiv z^e \pmod{N}$.

- Certainly, if Eve sends Bob the signature pair (w, z), he will recognize the signature as being valid: however, it is exceedingly unlikely that the message w (which will be some random eth power modulo N) will actually say anything meaningful.

- Ultimately, if we believe it is difficult to decrypt an arbitrary ciphertext that is encoded with Alice's public key, then it should also be difficult to forge Alice's RSA signature.

- Our next digital signature algorithm is based off of the ElGamal cryptosystem.
  - Alice first creates an ElGamal public key (p, a, b), where p is a large prime for which it is hard to compute discrete logarithms, a is a primitive root mod p, and $b \equiv a^d \pmod{p}$ for her secret choice d with $0 < d < p - 1$.
  - If Alice now wants to sign a message m, she first chooses a random integer k relatively prime to $p - 1$.
  - She then computes $r \equiv a^k \pmod{p}$ and $s \equiv k^{-1}(m - dr) \pmod{p - 1}$, and her signature is the triple (m, r, s).
  - If Bob wants to verify that Alice really signed the message m, he checks whether computes $b^r r^s$ is congruent to $a^m \pmod{p}$. If so, then he accepts the signature as valid, and if not he rejects it.
  - The procedure works because $b^r r^s \equiv (a^d)^r a^{ks} \equiv a^{dr} a^{m-dr} \equiv a^m \pmod{p}$.

- Suppose now that Eve has intercepted a message pair (m, r, s) that Alice has signed and wants to forge Alice's signature on a new message w.
  - Obviously, Eve cannot simply use the signature pair (w, r, s), since Bob will compute $b^r r^s \equiv a^m \not\equiv a^w \pmod{N}$ and reject the signature as invalid.
  - In order to find a valid signature z for her message w, she needs to find (r, s) that are solutions to the congruence $b^r r^s \equiv a^w \pmod{N}$.
  - If Eve picks a particular r and searches for s, she is attempting to solve $r^s \equiv a^w b^{-r} \pmod{N}$, which is equivalent to computing the discrete logarithm $\log_r(a^w b^{-r})$.
  - Another possibility is for Eve to try to choose the value of s first, but this requires solving an even more unusual congruence $b^r r^s \equiv a^w \pmod{N}$, which is a combination of a discrete-log and root-extraction problem.
  - It may be possible to choose r and s together in some more efficient manner, but it is not obvious how such a procedure would work.

- Ultimately, if we believe it is difficult to compute discrete logarithms modulo p, then it should also be difficult to force Alice's ElGamal signature.

- Unlike with RSA signatures, it is not easy to recover the original message m from the remaining portion of the signature (r, s) like it is with RSA signatures (where $m \equiv y^e \pmod{N}$).

- Like with the ElGamal cryptosystem, there are many legitimate ElGamal signatures associated to a particular message m.

- Eve's attack if Alice reuses the same secret value k also essentially carries through.
  - Explicitly, suppose Alice sends two messages m_1 and m_2 to Bob and reuses the same value of k for each message.
  - Eve would then intercept the two triples (m_1, r, s_1) and (m_2, r, s_2), where $r \equiv a^k \pmod{p}$, $s_1 \equiv k^{-1}(m_1 - dr) \pmod{p - 1}$, and $s_2 \equiv k^{-1}(m_2 - dr) \pmod{p - 1}$.
  - Then $s_1 - s_2 \equiv k^{-1}(m_1 - m_2) \pmod{p - 1}$, and since Eve knows each of s_1, s_2, m_1, m_2, she obtains a linear congruence modulo $p - 1$ for the value of k, namely, $(s_1 - s_2)k \equiv m_1 - m_2 \pmod{p - 1}$.
  - If $\gcd(s_1 - s_2, p - 1)$ is small (which would be expected since $s_1 - s_2$ is essentially random), then this congruence will have a small number of solutions, and Eve can try all of them until she finds the correct one.

- Once Eve has k, she can essentially compute d, since $rd \equiv m_1 - ks_1 \pmod{p-1}$. Again, since r is random it is likely that $\gcd(r, p-1)$ is small, and so Eve can write down all the possible values for d modulo $p-1$ and check for the one that has $b \equiv a^d \pmod{p}$.
- We will also remark that Eve can easily tell if Alice reused the same value of k by looking at the value of r in different signatures: if the values are the same then Alice picked the same k for each.

- In certain settings, Alice might want to be able to sign a message without knowing the contents of the message.
  - For example, suppose that Bob claims to have designed an algorithm for factoring any 10000-digit integer in five minutes.
  - He would like Alice to attach her signature to his algorithm so that, ten years from now when he wants to reveal his algorithm to the world (after giving sufficient time for everyone, but especially his bank, to change their cryptosystems), anyone else would then be able to verify that Alice signed a valid copy of his algorithm.
  - Of course, Bob does not want to reveal his secret algorithm to Alice: he wants Alice to create a blind signature, where she signs the message without knowing its contents.

- One procedure using a modification of the RSA signature is as follows:
  - Alice generates an RSA public key with $N = pq$ a product of two large primes and e relatively prime to $\varphi(N)$, and publishes (N, e).
  - Bob wants Alice to sign his secret message m. He chooses a random integer k modulo N and sends Alice the value $t \equiv k^e m \pmod{N}$.
  - Alice signs t by computing $s \equiv t^d \pmod{N}$ and sends s back to Bob.
  - Bob then computes $sk^{-1} \pmod{N}$: then (m, sk^{-1}) is a valid RSA signature pair.
  - Bob can then store the signature sk^{-1} until he is ready to reveal his secret message m, at which point anyone can use Alice's public key (N, e) to verify that $(sk^{-1})^e \equiv (t^d k^{-1})^e \equiv (k^{ed} m^d k^{-1})^e \equiv m^{de} \equiv m \pmod{N}$, thus verifying that (m, sk^{-1}) has been validly signed.

- Of course, some safeguards are needed: for example, in the blind RSA signature scheme above, Alice has no safeguard if, instead of attaching her signature to a secret factoring algorithm, Bob instead had her sign something else such as an agreement to volunteer for treason.
  - One basic way around this is to include an additional, unalterable piece of the signature that says something like "This is a blind signature and is not binding", or something of that sort.

- We remark that Alice should not use the same RSA public key both for encrypting messages and for making blind signatures as the two processes are essentially inverses.
  - Explicitly, suppose that Alice uses the key (N, e) to encrypt a message m to a ciphertext $c \equiv m^e \pmod{N}$ which Eve intercepts.
  - If Eve wants to find the original message m, she chooses an arbitrary k and asks Alice to sign $t \equiv k^e c \pmod{N}$.
  - Alice will return $s \equiv t^d \equiv k^{ed} c^d \equiv km \pmod{N}$, and then Eve can simply evaluate $sk^{-1} \equiv m$ to obtain Alice's plaintext.
  - The multiplication by k^e is not strictly necessary, but if Eve did not do this then Alice might notice that Eve is asking her to decode the ciphertext of a message she sent earlier. By multiplying by k^e, it is then impossible for Alice to detect that Eve is tricking her into decoding a message, because if k is randomly chosen then $k^e c$ will also be random.

4.1.2 Hash Functions

- One of the disadvantages of both the basic RSA and ElGamal signature schemes we have discussed is the fact that the signature is approximately the same length as the original message.
    - In general, it is rather inefficient if the signature on a document is the same length as the document itself: we would prefer to have the signature be much smaller.
- One way of doing this is to involve a hash function: it takes as input a message and returns a much smaller hash value of some fixed length (e.g., 64 bits) in a way that is deterministic but "unpredictable", in the sense that messages that are almost equal will usually have different hashes.
    - If Alice sends a message to Bob, she can then sign the hashed value of the message rather then the full message itself. Bob can then compute the hash of the message he received from Alice and verify that it has been correctly signed.
    - If the message had been altered, then the signature will not match the hash of the message Bob received, and so Bob will know that something has gone wrong.
    - Another application of hashing is for error detection: rather than signing a message, Alice could also send the hash of the original message along with the message itself. If the message's hash does not agree with the value Alice sent, Bob will know that he did not receive the message correctly.
    - Depending on how the hash is computed, Bob may even be able to pinpoint or even fix the error himself. Designing error-detecting / error-correcting procedures is a very interesting topic, but it is not one we will discuss further.
    - Instead, our interest is in using hashes for cryptographic purposes.
- Definition: A hash function H is a function that takes in messages of unbounded length and produces a hash value of fixed length (e.g., 64 bits).
- In order to use a hash function for cryptographic purposes, we would usually like H to satisfy some additional properties:
    - First, H should be easy to evaluate on any message m. (A typical assumption is that the evaluation can be done in time polynomial in the number of bits of m.)
    - Second, H should be hard to invert: for any y, it should be computationally infeasible to find a message x such that $H(x) = y$.
    - Finally, H should be "collision-resistant": it should be hard to find messages m_1 and m_2 such that $H(m_1) = H(m_2)$.
    - A weaker version of the third property is "weak collision resistance": for any message m, it is hard to find a message m_1 such that $H(m) = H(m_1)$. For many cryptographic purposes, this property is sufficient.
- The above properties are not terribly rigorous, and we will not make any attempt to rigorize them.
- It is difficult to give simple examples of hash functions that actually satisfy the three requirements above, although in practice there are several examples that seem to work well. (In fact, depending on certain conjectures in theoretical computer science, such functions may not even exist.) Here is a simple example of an insecure hashing function with an n-bit output that nevertheless captures some of the ideas:
    - Given a large message m, divide it into n-bit blocks (padding the last block with zeroes as necessary) b_1, b_2, ... , b_k.
    - Then define $H(m) = b_1 \oplus b_2 \oplus \cdots \oplus b_k$, where the operator $s \oplus t$ denotes the bitwise exclusive-or, so that $0 \oplus 0 = 1 \oplus 1 = 0$ and $1 \oplus 0 = 0 \oplus 1 = 1$ on bits. (On strings, therefore, we would have $101 \oplus 100 = 001$.)
    - We can see that H is easy to evaluate on any input, and also that if we change a random small number of bits of m, then $H(m)$ will also change by roughly that same number of bits. In particular, if there is a small number of transmission errors in receiving a message m, then the errors will change the value of $H(m)$.
    - Of course, it is very easy to invert this function H and to find collisions.

- To create better hash functions that are more "random", the typical procedure is to perform a number of additional bit-level operations (such as shifting, permuting bits, or multiplying by matrices) in multiple rounds to further randomize the results, in much the same way as block ciphers do.
    - In much the same way as symmetric block ciphers, there are many hash functions in current use, though it seems to be harder to design an effective cryptographic hash function than an effective block cipher.
    - One well-known series of hash functions is the Message-Digest (MD) family consisting of MD2, MD4, MD5, and MD6. The first three, developed in 1989, 1990, and 1991 respectively by Ronald Rivest were each widely used in common software before the hashing functions were broken (and, quite often, well after they were broken!).
    - Another well-known series is the NSA's Secure Hash Algorithm (SHA) family, consisting of SHA-1, SHA-2, and SHA-3. The first algorithm was published in 1995 and is still in current use, although theoretical attacks are currently possible with sufficient computing power, and several major internet browser developers have announced that they will phase out support for SHA-1 by 2017.
    - In general, the basic idea of many of these hashing functions is to create a "compression function" that takes two message blocks as input and returns a single block of output (in a way that seems essentially random). We then compute the hash by breaking m into blocks and then applying the compression function to pairs of blocks until a single block remains, which is the value $H(m)$.
    - The difficult part, naturally, is creating a good compression function: the goal is to find one that can be evaluated rapidly and which causes any change in any input bit to affect a large amount of the output in a way that is hard to predict.
- One idealized theoretical model for hash functions is called the "random oracle" model.
    - A random oracle takes in an arbitrary bit string and outputs a bit string of fixed length (say, 64 bits).
    - If the random oracle is given a new input value it has not been given before, it returns a random result (e.g., by flipping 64 coins).
    - If the random oracle is given a previously-seen input value, it returns the same result it did before.
    - From a theoretical perspective, a random oracle is uncomputable: it is not actually possible to create a random oracle using a finite amount of memory, since the oracle must store a potentially unbounded amount of information.
    - However, it is easy to see that an oracle function satisfies the requirements of the cryptographic hash function: it is easy to pass in values, but it is hard to invert and hard to find collisions.
    - The only method for inverting an oracle function would be to try different messages until an inverse image is found, and the only method for finding collisions would be to pass in messages until a collision is found.
    - When studying the security of signature and encryption protocols, it is often useful to study the scheme under the assumption that a random oracle is used for the hash function. If we can show that a protocol is secure with an oracle function (in the sense that we can show that breaking the system is equivalent to solving some hard problem such as integer factorization), that at least provides some evidence toward the security of the protocol for other hash functions that mimic randomness.
    - We should remark, however, that it has been shown that there exist theoretical cryptosystems that are secure when using a random oracle but are not secure for any concrete choice of hash function. (It is not clear whether this result actually says anything about hash functions in practice, but it should at least raise some concerns about how effective the random oracle model actually is.)

4.1.3 Hashing and Signing, Birthday Attacks

- For creating digital signatures, the key idea is that instead of signing the entire message itself, Alice instead signs a hashed value of the message.
    - If Alice uses a hashing function H whose output is (say) 256 bits, then by assumption it should be difficult for Eve to invert the function. Alice would then be able to sign the 256-bit hash of her potentially much larger message, meaning that her signature would be roughly 256 bits in length.

- Alice would then send Bob the pair $(m, \star)$ where $\star$ is the signature associated to $H(m)$. In order to verify the signature, Bob then checks whether $\star$ is a valid signature of $H(m)$.
- The hope is that this hashing-and-signing procedure will still provide adequate security if the hashing function and the signing algorithm are both secure and do not somehow interact in a way that leaks information.

- So suppose Alice signs the hash of a message $H(m)$, and Eve intercepts the result and wants to forge Alice's signature.
  - In order to attach Alice's signature to another message w, she needs the signature of $H(w)$ to be equal the signature of $H(m)$.
  - If the signature algorithm is one-to-one, then the only way this can occur is if $H(w) = H(m)$.
  - In other words, Eve must take w to be a preimage of m under the hashing function H. By the definition of H, it is assumed to be difficult to find such a w, and even if Eve could find such a w, it is unlikely that it would make any sense.

- A better way to try to forge Alice's signature of a hashed message is via a birthday attack, which we will illustrate with an example:
  - Alice wants to sign a legitimate contract with Mallory, a very lucrative employment contract that will pay her 5 million dollars per year.
  - Mallory would like to trick Alice into signing a different contract that only pays her 5 thousand dollars per year.
  - Alice's hashing function produces a 64-bit output, so she feels secure that Mallory will be unable to find a fraudulent version that has the same hash as her legitimate contract, since any other contract has a $2^{-64} \approx 10^{-19}$ probability of having the same hash as hers.
  - However, Mallory instead finds 35 places in the legitimate contract that a small change could be made (e.g., adding an extra space between words, swapping a word with a synonym, adding or removing a comma), and does the same for the fraudulent contract.
  - Thus, Mallory has 2^{35} different versions of each contract. Mallory's goal is then to find a collision between the hash of a contract on the first list and a contract on the second list. (Storing all the hashes for the first list requires a total of about $64 \cdot 2^{35}$ bits, or 2 terabytes: quite feasible.)
  - The probability that there is a collision between the hash of one contract on the first list with a contract on the second list is then approximately $1 - (1 - 2^{35}/2^{64})^{2^{35}}$, since the probability that the hash of any one contract on the first list does not match any contract on the second list is very close to $1 - 2^{35}/2^{64}$.
  - By taking logarithms and using the approximation $\ln(1 - x) \approx x$ for small x, we see that the probability that Mallory can find a collision is approximately $1 - e^{-64}$, which is very close to 1. Thus Mallory will obtain a variant m of the legitimate contract with the same hash as a variant M of the fraudulent contract.
  - Mallory then gives Alice the version m and asks her to sign it. Since $H(m) = H(M)$, Mallory then can claim that Alice actually signed the contract M when it is time to send out the paychecks.

- The ideal way to foil a potential attack of this nature is for Alice to make a small change to the document before signing it. In order to attach Alice's signature to a fraudulent version of the new contract, Mallory would then need to search through about 2^{64} different fraudulent versions of the contract before having good odds of finding one with the same hash (which would be almost impossible with current technology).

- In general, because of the feasibility of birthday attacks, it is generally accepted that one should use a hash function that is at least twice as long as the desired level of security.

4.1.4 Rabin Signatures

- We can also use hash functions directly in our signature algorithms. Here is a procedure for using a hash function that is based off of Rabin encryption:

- Alice first creates a Rabin modulus $N = pq$ that is the product of two large primes each congruent to 3 modulo 4.
- She also chooses a particular hashing function H that returns values modulo N, and publishes (N, H).
- To sign a message m, Alice makes random choices for the value u modulo N until she can find a solution x to the congruence $x^2 \equiv H(mu) \pmod{N}$. Alice's signature associated to the message m is then the triple (m, u, x).
- Alice can find an appropriate value of u rapidly: by the Chinese remainder theorem, it is sufficient for her to solve the congruence $x^2 \equiv H(mu)$ modulo p and modulo q.
- As with Rabin encryption, Alice can compute square roots modulo p and modulo q very quickly: specifically, if a is a unit then the congruence $x^2 \equiv a \pmod{p}$ has no solution if $a^{(p-1)/2} \not\equiv 1 \pmod{p}$, and otherwise the solution is $x \equiv a^{(p+1)/4}$.
- If Bob wants to verify Alice's signature, he checks whether $x^2 \equiv mu \pmod{N}$. If so, he accepts the signature, and if not he rejects it.

- Suppose now that Eve has intercepted a message pair (m, u, x) that Alice has signed and wants to forge Alice's signature on a new message w.
  - Obviously, Eve cannot simply use the signature pair (w, u, x), since Bob will compute $x^2 \equiv mu \not\equiv wu \pmod{N}$ and reject the signature as invalid.
  - In order to find a valid signature z for her message w, Eve needs to find a solution to the congruence $x^2 \equiv H(wu) \pmod{N}$, where she is free to choose x and u.
  - If Eve chooses x first, then she must solve $H(wu) \equiv x^2 \pmod{N}$. This is difficult because it is equivalent to computing the preimage of x^2 under the hash function H, which is assumed to be difficult.
  - If Eve chooses u first, then she must solve the congruence $x^2 \equiv H(wu) \pmod{N}$. This is equivalent to deciding whether $H(wu)$ is a square modulo N and (if so) extracting its square root: doing the latter is equivalent to factoring N from our analysis of Rabin encryption.
  - It is of course possible that there is some way of choosing x and u simultaneously, although this does not seem very likely.

- Ultimately, if we believe it is difficult to extract square roots modulo N (which is equivalent to factoring N), then it should also be difficult to force Alice's Rabin signature.

4.2 Zero-Knowledge Proofs, Basic Authentication

- One of the major flaws in the basic Diffie-Hellman key exchange is the lack of authentication of the participants. We would like to construct an identity-verification system wherein one participant can prove their identity to the other.
  - The two participants in such an identity-verification system are traditionally named Peggy (for "prover") and Victor (for "verifier").

- Peggy would like to have a way of proving her identity to Victor in a way that does not allow anyone else to impersonate her using the information. This is the fundamental idea of a zero-knowledge proof, which we first explain with an informal example[1]:
  - There is a cave with one entrance that is shaped like a circle. Inside the entrance are two paths labeled A and B – these paths both lead deep into the cave, to opposite sides of a door.
  - Peggy claims that she has discovered the magic words to open the door. Victor asks her to prove this, but Peggy does not want to reveal the secret to Victor.
  - They decide on the following method: Victor will wait outside the cave, and Peggy will proceed down one of the paths without letting Victor know which one. Victor will then enter the cave and shout either "A" or "B", at which point Peggy will return along that path.

[1] This example is adapted from a paper with perhaps one of the best paper titles ever: "How to Explain Zero-Knowledge Protocols to Your Children".

- Assuming Peggy knows the magic words, she can always pass the test: she simply walks back along the appropriate path (going through the door first, if she needs to).
- If Peggy does not know the magic words, then she only has a 50% chance of passing the test (because she can only leave via the path she went in), assuming that Victor chooses A or B at random. If Victor wants to be more certain that Peggy is telling the truth, he simply repeats the test as many times as he wants, until he is satisfied that Peggy does know the secret.
- Now suppose that Eve is watching and listening to all of the exchanges from outside the cave. From Eve's vantage point, it is entirely possible that Peggy and Victor could have colluded to fake all of the outcomes without Peggy knowing the magic words: Peggy and Victor could simply have agreed ahead of time which path Victor would call out, and then Peggy would take that path into the cave (so that she could exit without needing to go through the door). Thus, Eve cannot determine any information from the exchanges.

- The idea of a zero-knowledge proof was originally promulgated in a paper by Goldwasser, Micali, and Rackoff in 1985. (The paper was actually written in 1982 but was rejected from major conferences several times before finally appearing.)

4.2.1 A Rabin Zero-Knowledge System

- We now give an example of a zero-knowledge proof system modeled on the Rabin cryptosystem:
  - Peggy chooses two large primes p and q and publishes $N = pq$. She also chooses a secret s, computes $s^2 \pmod N$, and publishes this value. These two values serve as her identity.
  - Victor wants Peggy to prove that she knows the secret value s. They do this in the following way:
    * First, Peggy chooses a random unit u modulo N, and sends Victor the value of $u^2 \pmod N$.
    * Victor then chooses to ask for u or su at random.
    * Peggy sends Victor the quantity he requested.
    * If Victor asked for u, he squares the value and compares it to the value $u^2 \pmod N$ which Peggy sent earlier. If Victor asked for su, he squares the value and compares it to $s^2 \cdot u^2 \pmod N$, which he can also compute. If the appropriate value agrees, Peggy passes.
  - The challenges are repeated until Victor is satisfied that Peggy does indeed know the secret s.

- Example: Peggy chooses $N = 564481$ and $s = 53402$, and publishes the value of N along with the value $15592 \equiv s^2 \pmod N$.
  - Victor challenges Peggy to prove she knows s.
    * Peggy first chooses $u = 364210$ and sends Victor the message "404948", which is $u^2 \pmod N$.
    * Victor then flips a coin and sends Peggy the message "Send me su".
    * Peggy responds with the message "349565", which is $su \pmod N$.
    * Victor then checks whether $(su)^2 = 349565^2$ agrees with the values $s^2 \cdot u^2 = 15592 \cdot 404948$ that Victor already has. He indeed sees that $349565^2 \equiv 229231 \pmod N$, and $15592 \cdot 404948 \equiv 229231 \pmod{77}$ as well, so Peggy passes the first test.
    * Victor and Peggy then repeat the challenge 100 more times, and Peggy passes each time.
  - Eve attempts to impersonate Peggy. Victor challenges Eve to prove she knows s.
    * Eve knows $n = 564481$ and $s^2 \equiv 15592 \pmod N$, since these were published by Peggy, but Eve does not actually know s.
    * Eve guesses that Victor is going to ask for su. She chooses a random $x = 412009$, and then sends Victor the message "403457", which is equal to $x^2 \cdot (s^2)^{-1} \bmod N$. (She computes the inverse of s^2 using the Euclidean algorithm.)
    * Victor then flips a coin and sends Eve the message "Send me su".
    * Eve made her choice in such a way that she could respond to Victor's challenge if he asked for su: she responds with her value of x: "412009".

        * Victor verifies that $(su)^2 = 412009^2 \equiv 15592 \cdot 403457 = s^2 \cdot u^2$ modulo N. Eve passes this round.
    - Victor challenges Eve again.
        * This time, Eve guesses that Victor is going to ask for u.
        * She chooses a random unit $u = 116533$, and sends Victor the message "220672", which is $u^2 \pmod{N}$.
        * Victor then flips a coin and sends Eve the message "Send me su".
        * This time, Eve cannot respond to the challenge: she knows u (and could have responded to Victor's challenge if he had asked for u) but she does not know su, because she does not know s.
        * Thus, Eve fails the challenge, and now Victor knows she is an imposter.

- In order to verify that this protocol is a zero-knowledge proof, we must check three things.
- First, we need to check that the test is complete: that Peggy can always pass the test.
    - This is obvious, because Peggy knows both u and s.
- Second, we need to check that the test is sound: that if Eve does not actually know the value of s, that she cannot always pass the test.
    - In order to pass the test, Eve needs to be able to compute square roots of both $(su)^2$ and u^2 modulo n – but if she can do this, she can easily compute s, so this is equivalent to knowing s.
    - If Eve does not know s, then she can provide at most one of the two values u, su when challenged by Victor, so she has at most a $1/2$ probability of passing the test in this case.
- Finally, we need to check that the test is zero-knowledge: that Eve does not acquire any information about the secret s by observing real conversations between Peggy and Victor.
    - It is sufficient to show that Eve can simulate conversations with the same distribution as valid conversations between Peggy and Victor. If she can do this, then she gains no information by monitoring the actual conversations, because they are indistinguishable from fake conversations.
    - Eve simulates a challenge in the following way: first, she randomly decides whether "Victor" will ask for u or su.
    - In the first case, she has "Peggy" send the initial message $x^2 \pmod{N}$, where she chooses x at random – then the response from "Peggy" to "Victor's" challenge for a square root of x^2 is simply x (which Eve knows).
    - In the second case, she has "Peggy" send the initial message $x^2 s^{-2} \pmod{N}$, where she again chooses x at random – then the response from "Peggy" to "Victor's" challenge for a square root of $s^2(x^2 s^{-2})$ is just x again (which Eve knows).
    - In either case, this simulated conversation between "Peggy" and "Victor" is valid.
    - But now since Eve chose Victor's choice of u or su, along with the value of x randomly, the distribution of outcomes is the same as that of a real conversation.
- We will note that the strength of this identity-verification system depends on the difficulty of computing square roots modulo N, which is the same underlying problem that the Rabin encryption algorithm relies on, and which we proved was equivalent to factoring N.
- It is also very important to note that the above system is not a "proof of identity" in the way the term is colloquially used. The only information that is obtained, by anyone, following the zero-knowledge proof protocol is: Victor is now confident that the person he is communicating with knows the secret number s.
- This zero-knowledge proof system is easily adapted to create an authentication protocol using Rabin encryption. (For simplicity we will ignore the issues of nonuniqueness of square roots, but they can be dealt with.)

- Alice creates a public key N_a as in Rabin encryption, and Bob creates his own public key N_b. Alice and Bob need to be confident that these public keys were actually created by one another (which they could do, for example, by creating the keys in each other's presence), and that nobody else can factor their keys.
- Bob then wants to send Alice an encoded message m and makes her prove she can decode it, without revealing any other information. Bob follows Rabin encryption and sends Alice the value of m^2 modulo N_a.
- He and Alice then perform the zero-knowledge protocol described above to prove that Alice actually knows the secret value m (which she does, because she knows how to factor N_a and hence can compute the square root of m^2).
- Now Bob knows that his message reached someone who successfully decoded it, which (by assumption) can only be Alice.
- The procedure works in the other direction as well: Alice can send Bob an encrypted message and verify that it was decoded correctly by Bob.
- Alice and Bob can now continue sending messages to one another using their separate public keys (performing a zero-knowledge proof with each message), to be secure in the knowledge that the other party has received and decoded them.

- Now suppose Mallory is recording all of the conversations, or even attempting to impersonate Alice or Bob.
  - Whatever Mallory does, there is no way to interfere with the authentication procedure in either direction, because any change to any of the messages will cause the zero-knowledge proof to fail.
  - Mallory also cannot decrypt any of the encrypted messages, since that is equivalent to breaking Rabin encryption (which is assumed to be difficult).
  - At worst, Mallory could simply prevent messages from passing between Alice and Bob at all, but that is always possible over any communication channel.

- One of the drawbacks of the basic Rabin protocol is that it requires many rounds of communication between Peggy and Victor. This is not really an issue since it is very easy to parallelize the challenge rounds to do them all simultaneously, but introduces the extra cost of making the messages correspondingly longer.
  - Explicitly, to begin Peggy generates 100 different values u_1, u_2, ... , u_{100} and send all of their squares to Victor in a single message.
  - Victor then flips 100 coins and responds with the results, which indicate to Peggy which of u_i or su_i he wants.
  - Peggy then responds to the 100 challenges with the 100 corresponding values, and finally Victor checks all of the results. If they are all correct, then he believes Peggy knows the secret s, and if any of them is incorrect he rejects her claim. (Perhaps even to account for the occasional transmission error, Victor could decide only to reject if Peggy fails 5 or more of the 100 challenges.)

4.2.2 The Fiege-Fiat-Shamir Identification Scheme

- Another method for parallelizing the communication rounds in a zero-knowledge protocol is due to Fiege, Fiat, and Shamir:
  - Peggy chooses two large primes p and q and publishes $N = pq$. She also chooses k secret numbers s_1, s_2, ... , s_k, computes each of their squares mod N, and publishes these values. The set $(N, s_1^2, s_2^2, \ldots, s_k^2)$ serve as her identity.
  - Victor wants Peggy to prove that she knows the secret values s_1, s_2, ... , s_k. They do this in the following way:
    * First, Peggy chooses a random unit u modulo N, and sends Victor the value of $u^2 \pmod N$.
    * Victor chooses numbers $b_1, b_2, \ldots, b_k$ each of which is either 0 or 1, and sends them to Peggy.
    * Peggy then sends Victor the quantity $rs_1^{b_1} s_2^{b_2} \cdots s_k^{b_k} \pmod N$.

    - Victor then squares the quantity he received from Peggy and compares it to value of $r^2(s_1^2)^{b_1}(s_2^2)^{b_2}\cdots(s_k^2)^{b_k} \pmod N$ which he computes using the value r^2 that Peggy sent earlier along with the values $s_1^2, s_2^2, \ldots, s_k^2$ from her key.
  - The challenges are repeated until Victor is satisfied that Peggy does indeed know the secret numbers $s_1, s_2, ..., s_k$.

- The basic idea behind this version of the protocol is that there are 2^k possible quantities that Victor could ask for, and if Eve does not know any of the numbers $s_1, s_2, \ldots, s_k$, then the only way she could pass the test is if she guesses which one Victor is going to request.

  - Specifically, if Eve guesses that Victor will ask for $rs_1^{c_1}s_2^{c_2}\cdots s_k^{c_k}$, then she can generate a random value q and send Victor the value $q(s_1^2)^{-c_1}(s_2^2)^{-c_2}\cdots(s_k^2)^{-c_k}$ in the first step.
  - If Victor does ask for $rs_1^{c_1}s_2^{c_2}\cdots s_k^{c_k}$, then Eve will respond with the value q which will pass Victor's test.
  - However, if Victor instead asks for $rs_1^{b_1}s_2^{b_2}\cdots s_k^{b_k}$ for a different set of digits $b_1, b_2, \ldots, b_k$, then Eve would need to know the value $s_1^{c_1-b_1}s_2^{c_2-b_2}\cdots s_k^{c_k-b_k}$. If she does not know any of the s_i, then she almost certainly cannot compute this quantity since each of the exponents is either -1, 0, or 1 (and at least one of them is nonzero).
  - Thus, in one round of the protocol, the probability that Eve can pass the test is only $1/2^k$. Roughly speaking, passing one round of this test is the same as passing k rounds of the Rabin test.

4.3 Secret Sharing, Multiparty Computation

- We now turn our attention to "secret sharing" procedures and their applications to secured multiparty computation.

4.3.1 Shamir's Secret Sharing

- We begin by discussion secret sharing schemes, which are motivated by the following problem:

  - Alice is the manager of a bank, and she needs to distribute the secret code for the bank vault to her employees.
  - Alice does not trust any of her employees individually: if she simply gave the code to each of them, then any one of them could sneak into the vault and steal the money.
  - However, she is confident that no five or more of her employees would conspire together.
  - Alice would like a way to distribute the secret code among her employees in such a way that, if any four of her employees pool their information together they will not learn the code, but that any five employees can pool their information to obtain the code.

- There is a fairly simple way for Alice to create a system whereby the bank vault can only be opened by all of her employees working together:

  - If Alice has k employees, then she chooses a modulus N larger than the secret message m.
  - She then generates random residues r_1, r_2, ... , r_{k-1}, and gives the ith employee the value r_i for each $1 \le i \le k-1$. She then gives the last employee the value $m - r_1 - r_2 - \cdots - r_{k-1}$ modulo N.
  - The k employees can then recover the secret message m by adding all of their numbers together modulo N.
  - If any $k-1$ employees try to recover the message m, they will be unable to do so: the other employee's number could be any value modulo N, meaning that the desired sum m could be any value modulo N (with uniform probability).

- This method, however, cannot be easily extended to cover the case where Alice has 7 employees, any 5 of which she wants to be able to open the vault. (She could create a separate set of keys for each possible set of 5 employees, but that would already introduce an unreasonable amount of information.)

- In general, we refer to a system with k parties, any j of which can reconstruct a secret but such that no $j-1$ or fewer can, as a (j,k)-threshold scheme. Our goal is to create (j,k)-threshold schemes where $j < k$.
- Methods for creating more general (j,k)-threshold schemes where $j < k$ were first designed by Shamir and Blakely (independently) in 1979. We will discuss Shamir's method, which is easier to implement, but to do so we first require a result about polynomials modulo a prime:
- Proposition (Lagrange Interpolation): If p is a prime, then given any d points (x_1, y_1), ... , (x_d, y_d) having distinct x-coordinates modulo p, there is a unique polynomial $q(x)$ of degree $\leq d-1$ that passes through those d points modulo p.
  - This result also holds for real or complex polynomials (or more generally, polynomials over any field).
  - Proof: For existence, observe that the polynomial $q_i(x) = \dfrac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}$ of degree $d-1$ has the property that $q_i(x_j) = 0$ for $i \neq j$ and $p_i(x_i) = 1$, where we interpret the denominator as being the modular inverse of the product $\prod_{j \neq i}(x_i - x_j)$. (The inverse necessarily exists because p is prime and none of the terms is zero modulo p by hypothesis.)
  - Then the polynomial $q(x) = \sum_{i=1}^{d} y_i \cdot q_i(x)$ has the property that $q(x_k) = y_k$ for each $1 \leq k \leq d$, and $q(x)$ has degree $\leq d-1$.
  - For uniqueness, suppose $r(x)$ is another polynomial passing through those d points. Then $q(x) - r(x)$ is a polynomial of degree $\leq d-1$ that is zero at the d values x_1, ... , x_d.
  - But then $q(x) - r(x)$ must be divisible by $(x - x_1)(x - x_2) \cdots (x - x_d)$: since this polynomial has degree d, the only multiple having degree $\leq d-1$ is 0. Therefore, $q(x) - r(x) = 0$ so $q(x) = r(x)$.
- Here is Shamir's secret sharing procedure for creating a (j,k)-threshold scheme:
  - Alice chooses a prime p larger than the number k of her employees, and encodes her secret s as a residue class modulo p.
  - She also chooses the number j of her employees that she wants to be able to reconstruct the secret.
  - Alice chooses random elements $a_1, \ldots, a_{j-1}$ modulo p and constructs the polynomial $q(x) = s + a_1 x + a_2 x^2 + \cdots + a_{j-1} x^{j-1}$.
  - She then gives the value $q(1)$ to the first employee, the value $q(2)$ to the second employee, ... , and $q(k)$ to the kth employee.
  - By Lagrange interpolation, any j of her employees can pool their information together to generate the polynomial $q(x)$, since it has degree $j-1$ and is therefore the unique polynomial of that degree passing through their j points. Then they just compute the secret value $q(0) = s$.
  - On the other hand, if any $j-1$ of her employees pool their information, they still have no information about what the secret coefficient s is, because (again by Lagrange interpolation) there is a polynomial of degree $\leq j-1$ passing through their $j-1$ points and $(0, w)$ for any residue w.
  - Since the $j-1$ employees have no way to determine which of these polynomials is the correct one, they receive no information whatsoever about the value of $q(0) = s$.
- Example: Distribute the secret $s = 137$ among four employees so that any two of them can reconstruct the secret.
  - We choose $p = 227$. To construct our threshold scheme, we choose a random residue $a_1 = 154$ and then construct the secret polynomial $q(x) = 137 + 154x$.
  - The first employee receives $q(1) \equiv 64$, the second receives $q(2) \equiv 218$, the third receives $q(3) \equiv 145$, and the fourth receives $q(4) \equiv 72$, with all values reduced modulo p.
  - Suppose the first two employees want to reconstruct the secret. They use Lagrange interpolation and compute $q(x) = q(1) \cdot \dfrac{x-2}{1-2} + q(2) \cdot \dfrac{x-1}{2-1} = -64(x-2) + 218(x-1) = 154 - 90x \equiv 137 + 154x \pmod{227}$. Hence, they recover the secret $s = 137$.

- Or suppose employees 1 and 4 want to reconstruct the secret. They compute $q(x) = q(1) \cdot \frac{x-4}{1-4} + q(4) \cdot \frac{x-1}{4-1} = 64 \cdot (-3)^{-1}(x-4) + 72 \cdot 3^{-1}(x-1) \equiv 130(x-4) + 24(x-1) \equiv 137 + 154x$. Hence, they recover the secret $s = 137$.

- Shamir's secret sharing scheme is even more flexible than it might appear.
- For example, if Alice wants to make some of her employees more important than others (at least as far as getting into the bank vault goes), she can simply give them more shares of the secret.
    - For example, suppose Alice trusts her two most senior employees to be in the vault together, but doesn't want to allow any of the junior employees to access the vault unless there are three or more people.
    - Alice can then set up a threshold scheme requiring 4 shares to reveal the secret, and give each senior employee 2 shares each (i.e., two values of $q(x)$) and the remaining employees 1 share each. Opening the vault then requires either two senior employees can open the vault, one senior and two junior employees, or four junior employees.
- Furthermore, if Alice hires new employees and needs to allow them vault access, she does not need to create a new vault code: she can simply give the new employees additional values of $q(x)$, at least provided she hasn't run out of residues modulo p yet. (Of course, she should not give anyone the value $q(0)$, since that is the secret.)
- By combining threshold schemes together, it is also possible to deal with more complicated scenarios.
    - For example, imagine a high-security bank is storing an important high-stakes contract between Alice Industries and Bob Solutions. To ensure security and fairness, access to the vault storing the contract is only allowed when there are at least 5 Alice employees and 6 Bob employees present.
    - It would not be possible to give the employees of both companies shares of the same secret, since then all the Alice employees could pool their shares together to get into the vault (and similarly for the Bob employees).
    - Instead, we can combine several threshold schemes: first, we distribute the vault code between "Alice" and "Bob", and then we distribute the "Alice" secret among the Alice employees and the "Bob" secret among the Bob employees.
    - In the example above, if v is the vault code, then we could choose a random large prime p and a random residue $a \bmod p$: then the "Alice" shared secret is a and the "Bob" shared secret is $v - a \pmod p$. We would then distribute the secret a_1 among the Alice employees using the Shamir scheme with $j = 5$ and distribute $v - a$ among the Bob employees with $j = 6$.
    - Even if all the Alice employees pool their shares together, they can only recover their portion a, and will thus learn nothing about the actual vault code. Likewise, the Bob employees also cannot learn anything about v from their information.
- One drawback of the Shamir scheme as described above is that there is no way to correct for errors in communication, whether accidental or deliberate: any change to any of the pieces of information used to recover the secret will change the value obtained.
    - It would therefore be possible, in some cases, for a malicious employee, Mallory, to trick other employees into revealing their portions of a secret while sharing an incorrect value for their portion.
    - In principle, Mallory could then use the other revealed shares to find the secret, while the other employees would be unable to recover it (since they are still missing Mallory's piece). The other employees would at least know that something has gone awry when the secret-recovery process does not work correctly.

4.3.2 Secured Multiparty Computation

- We will now mention some applications of secret sharing in the area of secure multiparty computation.

- The idea of secure multiparty computation is that some parties (Alice, Bob, etc.) each have some private data that they wish to keep secret, but they want to design some way to pool their information together and evaluate some function on it whose result will be public, without revealing any of their private data.
    - One particular application of this idea is to create anonymous silent-bidding auctions: each participant puts in a bid for a particular item, and the highest bid wins the auction. However, the participants only want the winner to be identified along with their winning bid, and do not want anyone to know what any of the losing bid amounts were (since that knowledge is a trade secret).
    - Equivalently, in an auction, the participants wish to securely compute the maximum of their private bids without revealing any of the other bids.
- Here is the basic idea for one method of doing a multiparty computation: suppose Alice, Bob, and Carol want to add together their three secret numbers s_a, s_b, and s_c modulo p.
    - They ask two trusted authorities Trent and Arthur (whom they each trust not to tamper with the results) to help do the computations.
    - Alice, Bob, and Carol each choose random values r_a, r_b, and r_c modulo p, and send these values to Trent. They also send the values $s_a - r_a$, $s_b - r_b$, and $s_c - r_c$ (all evaluated modulod p) to Arthur.
    - Arthur and Trent then publish the sum of the three numbers they each received. Alice, Bob, and Carol can then evaluate their sum $s_a + s_b + s_c$ modulo p by adding together the numbers that Trent and Arthur computed.
    - No other information about Alice's, Bob's, or Carol's secret number is revealed by Trent or Arthur's number because of the way the secrets were split apart.
- To extend the above example, in the event that Alice, Bob, and Carol do not completely trust Trent and Arthur, they could ask additional people to do computations and use a secret sharing scheme to check the results once the computations are done.
    - For example, suppose Alice, Bob, and Carol ask Trent, Arthur, and Peter to help them add their secret numbers together.
    - Each of Alice, Bob, and Carol creates a Shamir polynomial $q(x) = s + rx$ for a randomly chosen residue r modulo p: they then send $q(1)$ to Trent, $q(2)$ to Arthur, and $q(3)$ to Peter (all results reduced modulo p of course).
    - Trent, Arthur, and Peter each sum the three values they receive and publish the results modulo p: these values are then $Q(1)$, $Q(2)$, and $Q(3)$, where $Q(x) = (s_a + s_b + s_c) + (r_a + r_b + r_c)x$ is the sum of Alice's, Bob's, and Carol's polynomials.
    - Alice, Bob, and Carol can then use any two of Trent's, Arthur's, and Peter's values to reconstruct the polynomial $Q(x)$, whose constant term is the desired sum of their secret numbers. They can do this for all three possible pairs and if any of the results differ, they will know that one of Trent, Arthur, and Peter has made an error (or tried to cheat!).
- In general, the difficult part of performing a multiparty computation is arranging the distribution of the information with the encryption method that allows for the desired computation to be done.
    - In the examples above, the Shamir encryption method commutes with addition: encryption and then adding produces the same result as adding and then encrypting.
    - It is much less obvious, for example, how one could use the Shamir method to compute a maximum, since the random values would change the relative order of the bets in an unpredictable way.
    - Ultimately, it is a nontrivial problem to determine how to compute various logical operations on distributed information in an efficient way, and multiparty computation has only been deployed on a wide scale rather recently.
- One of the first examples of a large-scale secured multiparty computation occurred in Denmark in 2008[2].

[2]See the paper "Multiparty Computation Goes Live" by P. Bogetoft et al., from the Cryptology ePrint Archive, Report 2008/068.

- To summarize: there is only one processor of sugar beets for the Danish market (Danisco), so each of the Danish sugar beet farmers needs to sell their crop to that company. The farmers want to get the highest price possible for their beets, while the processor wants to pay the lowest possible price.
- The two groups (the farmers and the processor) decided on a double-auction scheme to determine the "market clearing price": each seller specifies how much they are willing to sell at a number of various prices, and the buyer indicates how much they are willing to buy at each price. The price at which the total supply equals the total demand is then declared to be equal to the market price.
- The required computation was therefore to evaluate (i) the sum of all of the individual sellers' productions at each price, and then (ii) to find the price at which the total supply equals the total demand, without revealing any of the sellers' production functions or the buyer's demand function.
- The first portion of the computation could be performed using a Shamir-like procedure, but the second part is more difficult, as it requires comparing a number of values to one another to determine the largest. In actuality the procedure used was more complicated, and we will not give the specific details.
- Using a secured multiparty computation to determine the market price solved a number of logistical issues that would have existed using other methods: using a third party would have been very expensive, having the buyer compute the price would potentially allow them too much power given that they already have a monopoly, and having the sellers collectively set the price would not work since they are in principle competing with one another.
- Procedures like making the seller bids public but not attaching identities to them were also ruled out as having the potential to leak too much information to the buyers.

4.4 Digital Voting

- In this section we will outline a few schemes for digital voting.
  - There are various kinds of ballot schemes (ranked candidates, point systems, etc.) but we will restrict our attention to the simple case of an election between 2 candidates, where the candidate with the most votes wins the election.
- Ultimately, we would like a digital voting scheme to possess the following properties:
  - Each valid ballot must be from a voter who is legally allowed to vote, and each voter cannot vote more than once.
  - There should be no way for anyone to determine what candidate(s) any given voter chose to vote for.
  - After the election, each voter should be able to verify that their vote was counted properly without having to reveal their actual vote. (In fact, it is often desirable that voters cannot prove to anyone how they voted after they vote, to preclude the possibility of third parties buying peoples' votes.)
  - It should be possible to conduct a recount after the election is completed.
  - There should be redundancy in the system, so that errors in tabulation can be found and corrected.
- Most digital voting schemes actually used in practice are rather more cumbersome than we can discuss. Many of them are proprietary or closed-source, and many also rely on hardware security more than cryptography to function properly. We will therefore only discuss a few procedures for dealing with some of these requirements.

4.4.1 A Simple Shamir Voting Scheme

- The idea of trusted multiparty computing using the Shamir sharing scheme can also be used to create an extremely simplified digital voting system for an election with two candidates:
  - There are k voting authorities, and a "security parameter" $j \leq k$ is chosen: this is the minimum number of authorities who are trusted not to collude together to fix the results of the election.
  - A large prime p, much larger than the number of voters or voting authorities (at least 4 times as large), is also chosen, and the two candidates are arbitrarily assigned the labels $+1$ and -1.

- Each voter v then casts their secret "vote" as $s = +1$ or $s = -1$, and encodes this by creating their own polynomial $q_v(x) = s + a_1 x + a_2 x^2 + \cdots + a_{j-1} x^{j-1}$ for randomly chosen residues a_1, ... , a_{j-1} modulo p.
- The voter then sends $q_v(i)$ to the ith voting authority, for $1 \leq i \leq k$.
- The voting authorities sum all of the values they have received, and publish the result modulo p.
- Anyone can then use the results from any j of the voting authorities to reconstruct the vote total, by performing Lagrange interpolation to compute the sum polynomial $\sum_v q_v(x)$ (where the sum is taken over all voters): the constant term of this polynomial will give the total sum of the votes for all participants.
- If the result is a small positive result modulo p, then the candidate assigned the $+1$ is the winner, and if the result is a small negative result modulo p, then the candidate assigned the -1 is the winner. (If N is the total number of voters, then the results will lie in the range $[-N, N]$ modulo p, and so it will be easy to determine the actual vote totals.)
- The voting authorities cannot easily tamper with the results, because (if they did) the redundancy created by the other authorities would reveal that they have attempted to tamper with the outcome.
- The secured nature of the voting ensures that (even if an authority were to try to tamper with the vote totals) any alteration of the results will be detected, unless many of the authorities collude to make the same alterations.

- Of course, there are some drawbacks to the (extremely basic) system described above, as it does not follow most of the requirements we outlined above.
  - First, as described, the system has no authentication to make sure voters are not voting more than once, or are even legally allowed to vote at all.
- Another difficulty is that there is no error-correction mechanism: if there is an error at any stage of the computation by a voting authority, in even one single vote, the end result will be incorrect and they will have no way to correct the error.
  - If the authorities are tallying ten million individual votes, the odds that every vote is cast correctly with no errors seems like it is probably low.
  - One way to fix this is to break the computations into smaller pieces (e.g., by tabulating results in each voting precinct separately), lowering the chance of a catastrophic error at any stage.
  - Another way would be to include some error-correcting mechanism into the votes themselves (e.g., by adding redundancies in the information transmitted).
- A more serious problem is that any individual voter cannot verify that their vote was actually counted.
  - They do not have access to the voting authorities' internal totals, and after the election is over, there is no way for anyone to reconstruct any individual person's vote.
  - If somehow their vote was accidentally (or maliciously) discarded by all of the voting authorities, the voter would have no way to prove that their vote was not counted.
  - One way to try to partially address this problem would be for each voting authority to create a secure key, and then perform a zero-knowledge authentication with each voter: this way, each voter would at least be assured that their vote was received by the voting authority.
- The system is also vulnerable to malicious votes.
  - If a voter instead encodes their polynomial with a constant term of $+20$ or -20 rather than $+1$ or -1, then their vote will be counted as being worth 20 times as much as it should.
  - There is also no way for the voting authorities to detect this behavior, because the votes are distributed and the authorities are assumed not to collude to decode votes.
  - Another problem would be if a malicious voter sends different results to each of the voting authorities, in an attempt to cause the results to be undecodeable at the end of the computation. (Since the results will no longer agree, it would appear as if each voting authority tried to tamper with the outcome of the election.)

4.4.2 A Blind-Signature Voting Authentication Scheme

- We can create an authentication method for digital voting using the ideas behind blind signatures. Here is the basic idea:
  - Alice is the voting registrar, and is responsible for certifying that any individual voter is legally allowed to vote and making sure that nobody votes twice.
  - Bob wishes to vote. To do this, he fills out his ballot and places it in a sealed envelope containing a sheet of carbon paper (that can be written on by someone without opening the envelope). Bob then encloses his ballot in another envelope containing his identity information.
  - Alice receives Bob's sealed envelope and verifies his identity. She opens the outer envelope, signs the carbon paper without opening Bob's sealed ballot envelope, and returns it to Bob.
  - Bob then anonymously mails his sealed ballot to the voting tallier Trent, who opens it and checks that it has been signed by Alice.
  - Alice never learns Bob's vote, because Bob can see that his envelope was never opened, and Trent never learns Bob's identity, because that information was removed after Alice verified Bob's identity.
- To perform each of the above steps digitally, we require a few ingredients:
  - Alice must be able to verify Bob's identity in a way that does not allow anyone else to impersonate him.
  - Bob must either be able to encrypt his ballot in a way in a way that Alice cannot decode, but on which she can put a blind signature that Trent can verify, but which is hard to forge.
  - Bob must be able to verify Trent's identity, and send in his ballot along with Alice' signature in a way that allows him to verify that Trent received it.
- Each of these steps can be achieved using the procedures we have already developed. Here is one method:
  - Alice and Bob perform zero-knowledge authentications to establish their identities with one another.
  - Bob encrypts his ballot m using his preferred encryption method E_b (e.g., RSA) and sends his encrypted ballot $E_b(m)$ to Alice. (The encrypted ballot is in turn encrypted using Alice's public key, to prevent Eve from obtaining Bob's encrypted ballot.)
  - Alice verifies Bob's identity and checks him off the voter rolls. She then puts a blind signature on Bob's encrypted ballot $E_b(m)$ and sends back her signature $S(m)$.
  - Bob then checks that Alice has correctly signed his ballot. He then anonymously sends the pair $(m, S(m))$ to Trent.
  - Trent next authenticates himself to Bob (without requiring Bob to verify his identity). Bob then sends his ballot to Trent, and then Trent performs a zero-knowledge proof to demonstrate he can decode Bob's ballot.
  - To prevent signature forgery, Trent could also require each ballot to be padded with a random 512-bit string: this would prevent Eve from being able to encrypt all possible ballots using Trent's public key and compare them to the ballots sent by everyone.
  - To prevent multiple voting, Trent could ask Alice to append a random 512-bit string to each signature that she generates using each voter's identity (so that each string is tied to a given voter, but not in a way anyone else can identify). It would then be easy for Trent to determine whether anyone has tried to vote more than once, since the associated 512-bit strings on their votes would be the same.

4.4.3 Bingo Voting

- A voting scheme developed by Bohli, Müller-Quade, and Röhrich[3] known as bingo voting involves using random numbers. Here is the basic idea:

[3] See the paper "Bingo Voting: Secure and coercion-free voting using a trusted random number generator" by J.-M. Bohli, J. Mueller-Quade, and S. Roehrich, from the Cryptology ePrint Archive, Report 2007/162.

- Each of the n candidates has an urn containing a large pile of numbered stones, where there are as many stones as possible voters, and all of the stone numbers in all the urns are different.
- To vote, each voter secretly removes a stone from $n-1$ of the urns (they skip the one associated to the candidate they want to vote for), marks it, and writes down the number on that stone. If the stone they pick up is already marked, they choose another one.
- They also write down a random number they choose, and turn the ballot consisting of their n numbers.
- To tally the votes, the voting authority reads through each ballot and removes the appropriate stones from their respective urns. (They can easily verify that an individual ballot is valid by making sure they do not take 2 stones from the same urn.)
- The winner is then the candidate whose urn has the most stones remaining.

- The only issue with the above procedure (aside from the possibility of accidentally choosing a random number that happens to be associated to a stone) is that the voting authority cannot publish all of the results afterwards, since it would reveal each person's ballot.

- By using encryption, we can modify the procedure to create a paper trail that allows each voter to verify that their ballot was counted without revealing whom they voted for:

  - For each candidate in the election, a pool of N labeled "dummy numbers" is created (as many numbers than there are voters) modulo M, where M is much larger than N. This pool is kept private.
  - Each dummy number is also encrypted with a public encryption function that is hard to invert but easy to verify. This list is published along with a zero-knowledge proof that the dummy numbers are equally distributed among the candidates.
  - To fill out a ballot, a voter decides on their candidate and generates a large random number modulo M.
  - The central voting machine then gives them one dummy number from each of the candidates they did not vote for, and prints out a ballot receipt consisting of a number associated to each candidate: their random number for their chosen candidate, and the dummy numbers for the others. (They can verify their ballot is correct by checking that the random number they chose is the one attached to their candidate, and that the other numbers encrypt to dummy numbers.)
  - The central voting machine keeps track of which dummy numbers have been used and which have not, so as not to give out the same number twice.
  - Each ballot is then published, all of the unused dummy numbers for each candidate are also published, and the identifications of which encrypted dummy numbers are associated to each candidate are published. The winner is the candidate who had the most unused numbers.

- Each individual voter can then verify that their ballot was counted and that they voted for the correct candidate, since they can identify their ballot on the list of published ballots and they can check that their encrypted dummy numbers are associated to the candidates they did not vote for, in the correct manner.

  - This would prevent the central election authority from changing the candidates that the dummy numbers were associated to during the election.

- To check that the election was properly conducted, a few things need to be verified.

  - First, it must be checked that none of the "unused" dummy numbers was actually used on a ballot. This is easy to do by encrypting all of the unused dummy numbers and verifying that none of them appeared.
  - Second, it must be checked that no dummy number was used twice. This is easy to do simply by searching through all the ballots and looking for duplicates.
  - Third, it must be checked that each ballot contains exactly one non-dummy number and that each of the unrevealed dummy numbers was used on only one ballot. These can both be accomplished using a zero-knowledge protocol without revealing any additional information (this is not entirely trivial, and we will not discuss the details).

- The only potential issue is that a voter could accidentally choose a random number that is equal to one of the dummy numbers for some candidate.

- This can be made extremely unlikely by choosing the size of the random number to be much larger than any of the dummy numbers (e.g., a factor of 2^{64} larger).
- As an extra security step, the voting machine could perform a zero-knowledge proof on the pool of dummy numbers to make sure that the random number chosen by the voter does not appear on the list (again, without revealing the dummy number itself). In the unlikely event that it does, it could simply ask the voter to choose a new random number.

- We will also remark that this type of system has the advantage that the only practical difficulty during voting is creating random numbers in the voting booth.
  - The original proposal was to use a random number generator similar to a "bingo cage" in the voting booth itself to create the random numbers used during voting (whence the name of the voting system).
  - In this way, voters will feel that their random numbers are actually secure, since they choose it by themselves.
  - All of the time-consuming random-number generation (the generation and encrypting of the dummy numbers) is done before the election, and the remaining large computation of certifying the election results as valid occurs after voting is finished.
  - The only practical issue is in trying to manage the sizes of the numbers involved, both so that the ballot receipts are manageable and so that it is feasible to search through and do the necessary computations with the election results.

4.5 Digital Cash and Digital Currency

- In this section we will outline some digital cash and digital currency systems.
  - By digital cash we mean a digital equivalent of an existing currency (such as euros or US dollars) whose equivalence is backed by an issuing authority such as a bank or government.
  - By digital currency, we mean a new currency that is traded in digital form, whose value may be pegged to a particular existing currency or whose value may "float" (with a variable exchange rate).

- One of the advantages of physical cash is that it is essentially anonymous, in contrast to most digital payment systems.
  - If Alice has a pile of paper money she wishes to spend at Bob's shop, she can give Bob the money and purchase her items without having to reveal her identity to Bob or anyone else.
  - In contrast, if Alice wishes to purchase an item from Bob's shop using a credit card, her card issuer will know where she is making her purchase and how much money she is spending.
  - Furthermore, Bob also knows Alice's identity, since it is necessary to verify her identity to make sure the credit card is not stolen, and he knows which bank she uses (since he will receive payment from the bank).

- To create an effective digital cash that is equivalent to paper cash, we would like to satisfy the following requirements:
  - The cash can be sent electronically and divided into smaller pieces, but cannot be easily copied.
  - The cash can be transferred from one person to another.
  - Spenders can remain anonymous during legitimate transactions: recipients cannot determine the identity of spenders, nor can any third parties (e.g., the recipient's bank).
  - A spender can transfer cash to a recipient without requiring immediate communication with a third party.

- The fundamental difficulty in creating a digital cash (or currency) is allowing for anonymity while preventing counterfeiting.
  - Imagine Alice has a digital coin she wants to spend in Bob's shop.

- It is essentially trivial to copy digital information, so any digital coin must be linked to its owner: otherwise, anyone who obtains the underlying information in the coin would be able to spend it.
- For example, if the coin contains no information linking it to its owner, Alice could spend one copy of her coin in Bob's shop and another in Carol's shop. Bob and Carol would like to prevent Alice from doing this when she makes her purchase. But if the transactions are completely anonymous and the coin is not linked to Alice, they will have no way of doing this: all Bob and Carol will know is that they have each received identical copies of the same coin with no way to trace the double-spending back to Alice.
- Thus, coins must be linked to their owners, in a way that does not easily identify their owners if the coins are spent legitimately: Alice then simply needs to prove she owns her coin to Bob when she wants to spend it in his shop.
- We can achieve this with a "restricted digital signature": a digital signature that ties Alice to her coin that cannot be easily modified or decrypted, and which is itself encrypted in such a way that Alice's identity is not easy to determine.
- In order for Alice to spend a coin, she must also transfer ownership of the coin to Bob, so we also require the signature scheme to be flexible enough to allow Alice to sign over ownership of a coin to Bob (so that he can now legitimately claim ownership of it and Alice cannot) without allowing Bob to take ownership of her other coins.

4.5.1 Digital Cash With A Central Bank

- We will outline a pair of digital cash schemes involving a central bank.
  - The participants are Alice (who wants to spend her money in the store), Bob (the store owner), and the central bank where Alice and Bob keep their money and which backs the digital cash.
  - For simplicity in our discussion, we will declare that all coins are worth 1 dollar and that Alice wants to purchase an item worth 1 dollar.
  - First, Alice will tell the bank to withdraw 1 dollar from her account and create a digital coin for her to use that is digitally (but anonymously) signed to her.
  - Next, Alice will spend the coin in Bob's shop and transfer the ownership of the coin so that it now belongs to him.
  - Finally, Bob will return the coin to the bank, which will verify that he is the correct owner and credit 1 dollar to his account.
  - The bank will not know that the coin Bob is depositing in his account originally came from Alice: all it knows is that it has properly signed this coin and that nobody else has attempted to deposit the coin in their account.
- Here is a simple scheme to accomplish some of these goals:
  - Alice's bank first publishes a blind signature key.
  - To withdraw a dollar, Alice generates a coin c, applies a public hash function H, and then encrypts it with her encryption function E_a.
  - She sends the encrypted coin information to her bank, which puts a blind signature $s(H(c))$ on $H(c)$ and returns it to Alice, while also marking that she has withdrawn 1 dollar from her account.
  - Alice verifies the signature is correct. She now has the bank's signature $s(H(c))$ on her coin which she can use to demonstrate to Bob that the coin is backed by the bank.
  - To spend her coin, she gives Bob the value c along with the signed hash $s(H(c))$.
  - Bob verifies that the signature is correct, and (assuming it is) accepts the coin. He then sends c and $s(H(c))$ to the bank, which verifies that the coin is correctly signed and then credits Bob with 1 dollar to his account.
  - The bank also marks down the coin c as having been deposited, so that Alice (or Bob) cannot attempt to deposit the coin again.

- This scheme is anonymous and does prevent double-spending, but it suffers from the flaw that there is no way to punish double spenders.
    - Since the bank puts a blind signature on Alice's coin, and its signature function and hash function are hard to forge, it knows that the coin Bob is trying to deposit was created legitimately and must have originated from someone's account. However, it will not be able to identify the coin as Alice's when Bob deposits it, since it put a blind signature on her coin.
    - Likewise, Bob learns nothing about Alice's identity when she transfers a coin to him, since Alice generates the coin c herself however she wants, and the bank does not include any of Alice's identity information in the coin itself.
    - It is also not possible to spend a coin twice, since that would be detected by the bank when someone attempts to deposite the duplicate coin.
    - However, in the event that two different people try to deposit the same coin in the bank, there is no way to know who is at fault and thus no way to decide who the rightful owner of the coin is. For example, suppose Alice and Bob both try to deposit the same coin.
    - It could be that Alice copied her coin, spent it at Bob's shop, then immediately re-deposited it while hoping Bob would wait to deposit his copy. (In this case, Alice was trying to cheat.)
    - Or it could be that Bob deposits the coin Alice gave him, and then he copies it and spends the copy in Alice's shop later. (In this case, Bob was trying to cheat.)
    - Or it could be that Mallory created the coin, visited both of their shops, and spent a copy of the coin in each of them. (In this case, neither Alice nor Bob was trying to cheat.)
- One way to address the double-spending issue is for Bob to contact the bank immediately upon receiving a coin and requesting to deposit it. If the bank says the coin has already been deposited, then Bob rejects Alice's transaction. However, this violates one of the desired conditions that transactions can be performed without immediate interaction with a third party.
- A better way around this issue was developed by David Chaum, Amos Fiat, and Moni Naor[4], who modified the general protocol above to provide a way to identify and punish double-spenders. Here is the general procedure for the setup:
    - First, the bank decides on a "security level" k which will determine how difficult it is to get away with double-spending. (The probability of catching a cheater will be roughly $1 - 2^{-k/2}$.)
    - The bank also chooses a large RSA public signature modulus N (we will use $e = 3$ for simplicity) and specifies two hash functions $f(x, y)$ and $g(x, y)$ each taking two inputs and returning a single value modulo N.
    - Alice sets up an account numbered u with the bank, and the bank also keeps track of an associated counter v.
- To create a coin, Alice does the following:
    - First, she chooses k quadruples (a_i, c_i, d_i, r_i) modulo N which she will use to create her coin.
    - She then computes the k "blinded values" $B_i = r_i^3 \cdot f(x_i, y_i) \bmod N$ for each $1 \leq i \leq k$, where $x_i = g(a_i, c_i)$ and $y_i = g(a_i \oplus (u||(v + i)), d_i)$, where $\oplus$ denotes bitwise exclusive or and $||$ denotes concatenation.
    - The bank then chooses a random subset R of $k/2$ values of i in the range $\{1, \cdots, k\}$ and Alice sends the values (r_i, a_i, c_i, d_i) to the bank.
    - The bank then verifies that these quadruples yield the blinded values B_i Alice sent earlier: it can do this because it knows u and $v + i$ and thus can compute x_i and y_i from Alice's values r_i, a_i, c_i, d_i.
    - If the verification works, the bank then signs the product of the blinded values that Alice did not reveal by sending her the value $t = \prod_{i \notin R} B_i^{1/3} \pmod N$.
    - Alice verifies this signature and then uses the result to compute her coin $C = \prod_{i \notin R} f(x_i, y_i)$ by evaluating $t \cdot \prod_{i \notin R} r_i^{-1}$. She and the bank also increment Alice's counter by k.

[4]See the paper "Untraceable Electronic Cash" by D. Chaum, A. Fiat, and M. Naor from Proceedings on Advances in Cryptology, 1990.

- Finally, Alice reorders her quadruples so that the first $k/2$ of them are the ones that the bank did not request.
- Note of course that Alice cannot easily forge the bank's signature to create the coin, since she would have to break their signature algorithm.

- Now that Alice has a signed coin, she can then use it to pay Bob as follows:
  - Alice sends her coin C to Bob.
  - Bob sends Alice a random binary string of length $k/2$.
  - If the ith bit of the string is 1, then Alice sends Bob the values a_i, c_i, y_i. If the ith bit is 0, then Alice sends x_i, $a_i \oplus (u||(v+i))$, and d_i.
  - Bob then verifies that these pieces of data fit the value C provided: in the first case he evaluates $f(x_i, y_i) = f(g(a_i, c_i), y_i)$ and in the second case he evaluates $f(x_i, y_i) = f(x_i, g(a_i \oplus (u||(v+i)), d_i))$.
  - He then checks that C is equal to the product of these values.
  - To deposit the coin, Bob sends C along with Alice's responses to the bank, which verifies everything is accurate and then credits his account.

- The scheme remains anonymous on both ends:
  - The only identifying information related to Alice is her bank account number u.
  - But this information is always used in the form $a_i \oplus (u||(v+i))$, and since a_i is a random string, it appears random. Furthermore, Alice does not reveal both a_i and $a_i \oplus (u||(v+i))$ in any step, so Bob cannot ever extract u from the information Alice provides.
  - If f happened to be a weak hash function and Bob was able to invert it, he would still have to search through most of the possible input pairs (a_i, c_i) or $(a_i \oplus (u||(v+i)), d_i)$ to try to extract Alice's identity u, since he is only given one of the values c_i or d_i. (This is why the random parameters c_i and d_i are included.)
  - Likewise, the bank will not be able to determine that the coin that Bob requests to be deposited came from Alice, since the quadruples (a_i, c_i, d_i, r_i) Alice gave the bank when the coin was created do not overlap at all with the information that Bob gives the bank, since that was drawn from the other $k/2$ indices i.
  - All the bank will know is that Bob sends them a quantity that is signed using their RSA key (N, e) where $e = 3$.

- The key property of this system is that if Alice tries to double-spend a coin, then the bank will be able to catch her with high probability, and even identify which coin it was:
  - The bank compares the two transcripts where Alice spent the same coin and looks for a place where the ith bit was a 0 in one challenge string and a 1 in the other. If the strings were randomly generated then since they have length $k/2$, the probability that this occurs at least once is $1 - 2^{-k/2}$.
  - The parts of the two corresponding transcripts will then reveal Alice's values (a_i, c_i, y_i) and $(x_i, a_i \oplus (u||(v+i)), d_i)$.
  - The bank can then compute the exclusive-or $a_i \oplus [a_i \oplus (u||(v+i))] = u||(v+i)$ to obtain the bank account number u (identifying Alice) along with the counter value v (identifying which coin it was).

- One issue is that, while the bank will know that Alice tried to double-spend a coin, it cannot actually prove this fact to anyone else, since the bank can easily generate forged transcripts to frame her. This issue can be fixed by adding an extra level to the procedure by having Alice append a random string to the account number she uses in the computation, whose signature she provides to the bank.
  - Explicitly, Alice generates two random integers (w_i, z_i) for each i, and then replaces u with $u||w_i||z_i$ in all of the above computations.
  - She then posts her own public signature key, and signs the value $g(w_1, z_1)||g(w_2, z_2)||\cdots||g(w_k, z_k)$ and gives it to the bank as part of the coin-creation procedure.

- Then, when the bank makes its choice, Alice provides the associated $k/2$ pairs (w_i, z_i) to the bank as well.
- If Alice double-spends a coin, then the bank can recover at least one additional pair (w_i, z_i): its set of $(k/2)+1$ pairs (w_i, z_i) is then taken as proof that Alice cheated.
- The bank cannot generate these pairs on its own, and it cannot create forged transcripts because that would require forging Alice's signature.
- Even if the bank is able to invert the hash function g, Alice only needs to reveal all of her pairs (w_i, z_i) to reveal that g was inverted, since it is very unlikely the bank computed the same inputs to g that she used.

- We will remark that, as interesting as these examples are on a theoretical level, none of these types of digital cash systems are still in use.

4.5.2 Cryptocurrencies, Bitcoin

- Since 2008, a number of "cryptocurrencies" (a digital currency using cryptography to secure transactions and create new currency units) have been designed and promulgated.
- The most well-known example of a cryptocurrency[5] is probably bitcoin, whose design was first published in 2008 in a paper by "Satoshi Nakamoto". (It is not known whether this name is a pseudonym, or even whether the creator was a single person or a group of people.)
  - The most basic difference between bitcoin and the other digital currencies we have discussed is that bitcoin relies on a distributed peer-to-peer network rather than a centralized bank.
  - Transactions are verified by individual members in the network and recorded in a public ledger called the block chain, in such a way that it would be extremely computationally expensive to forge a transaction or change any part of the result other than through legitimate exchanges.
  - The advantage of this type of system is that it does not require a centralized exchange at a bank, but merely anonymized contact with another member of a network to record transactions after the fact. The disadvantage is that the only backing bitcoin has is from the network itself: unlike (for example) the digital cash systems we have described above, bitcoin is not denominated in an existing currency whose value is backed by a large institution such as a government, nor are any exchanges from bitcoin to a traditional currency backed by anything other than a decentralized market.
  - There are a number of legal issues that have yet to be resolved regarding cryptocurrencies, one of the most obvious of which is the fact that the anonymous and untraceable nature of the transactions means that it would be comparatively easy to use them for money laundering or tax evasion.
  - Likewise, cryptocurrencies are often a natural choice for purchasing of illegal goods since it is also essentially impossible to trace such purchases (again due to the anonymous nature of the exchanges).
  - It is also essentially impossible to catch anyone who steals units of a cryptocurrency from another legitimate owner because of the anonymous nature of all transactions. In February 2014, for example, the "Mt. Gox" bitcoin exchange based in Tokyo, which at the time handled a significant majority of all bitcoin transactions worldwide, suspended trading and filed for bankruptcy protection because approximately 800000 bitcoins (valued at over $400 million at the time) had been stolen directly from the bank's accounts.
  - Another concern is the resources necessary to implement cryptocurrency creation: in order to make the system difficult to forge, a typical strategy is to require new transactions to come with a "proof of work": a difficult computation ("work") whose time investment can be easily verified by anyone else ("proof of work"). To make the system effective, the computations must consume enough real-world resources (computing time, electricity) that would make it unreasonably expensive to break the system.

- Here is a rough outline of the general setup of bitcoin[6]:

[5] Legally speaking, it is not currently resolved in 2016 whether bitcoin is actually a "currency", but for consistency with the rest of our discussion we will ignore this semantic question.

[6] For slightly more detail, see the 2008 publication "Bitcoin: A Peer-to-Peer Electronic Cash System" from the author "Satoshi Nakamoto" of bitcoin.org.

- An electronic coin is defined as a chain of timestamped digital signatures, and Alice's ownership of a coin is indicated by having her signature be the most recent one attached to the coin. To transfer ownership, Bob can first verify that Alice' signature is the most recent one attached to her coin, and then Alice will provide Bob with sufficient information to allow him to sign the coin. All of the signatures are hashed so that they are essentially impossible to forge.
- To prevent Alice from being able to double-spend a coin, rather than using an individual central authority (e.g., the bank) that keeps track of whether particular coins have been spent, the idea is instead that all transactions are publicly announced (in an anonymized manner that does not reveal the identity of Alice or Bob) and incorporated into a public ledger (the block chain) that is difficult to forge, from which all participants then agree on a single history of all transactions.
- Bob then uses the network to check whether Alice's coin has already been spent according to the coin's history. If not, he accepts her transaction and announces to the network that he now owns the coin (again, in a way that does not reveal him) and the transaction is added to the block chain.
- To implement this protocol in a way that does not allow anyone to forge transactions, the idea is to require "proof of work" in order to add new information to the block chain: one must perform a difficult computation ("work") whose the time investment can be easily verified by anyone else ("proof of work").
- In the case of bitcoin, the required computation is to find a string that when appended to the transaction information yields a SHA-256 hash that begins with a specified number of zeroes. In essence, the requirement is to partially invert a hash function that is believed to be fairly secure: the only obvious way to do this is via a brute force search, so creating a hash that meets a target of k zeroes requires roughly 2^k computations.
- Each block in the chain is hashed with the ones before it, so once a block is incorporated into the chain, changing it would also require changing all of the subsequent blocks.

- The network is created in the following manner:
  - Each new transaction is publicly announced to all nodes, and each node collects new transactions into a block that it will attempt to add to its chain.
  - Each node then works on finding a proof-of-work for its block. When it does, it announces the block to all other nodes.
  - Each node accepts a received block only if all its transactions are valid and have not already occurred, and its proof-of-work is valid.
  - Nodes indicate their acceptance of a block by incorporating it into their chain and working to create the next block in the chain, using the hash of the accepted block as the previous hash.

- Nodes should always consider the longest chain to be the correct one, and work to extend it. (This longest chain is the "accepted public ledger" of all transactions.)
  - In the event that two nodes simultaneously generate different versions of a new block, some nodes may accept one version and some may accept the other depending on the order in which the blocks are received.
  - In this case, each node will attempt to extend one of two versions of the chain: one of these forks will eventually generate a new block, at which point it will become the longest and all other nodes will switch to it.
  - Due to the essentially random nature of successful block creation (a proof-of-work claim will succeed essentially at random, independent of the data in the block itself), the blocks that are backed by the most CPU power will be incorporated into the chain the fastest, and thus with high probability they will become the universally accepted chain since the members of the network collectively agree to accept the longest chain.
  - As long as each new transaction reaches enough nodes, it will eventually be incorporated into a block that will become part of the chain and thus be recorded in the public ledger. To incentivize nodes to include a particular transaction, it is also possible to include a transaction fee: Alice could agree to transfer 1.04 bitcoins out of her account, 1.02 of which go to Bob and 0.02 of which go to whomever successfully creates the block containing her transaction record.

- In principle, if a new collective with more computational power than the current network decided to create a new chain, it would be possible to create a new forged chain (or portion of one) that would overtake the "legitimate" existing chain. However, this would require forging all the results in the current chain before modifications could be made to future blocks: thus, the older a transaction is, the harder it becomes to alter the public ledger to change it.
- To counteract this possibility, the first transaction in a block is a special transaction that creates a new coin owned by the creator of the block. This provides an incentive for nodes to support the current network as much as they can: if they create a block that is incorporated into the chain, they get a monetary reward, while if their block is eventually rejected then they get nothing.
- Everything is, in some sense, provisional: if in the future a sufficiently large network did succeed in successfully forging a new block chain that is longer than the currently existing one, all of the existing transactions and coin transfers in the old chain would effectively cease to exist. However, this is really an economic problem, and it is not one specific to bitcoin: any currency would lose its value if a majority of parties using it no longer believed it was worth anything.

- In order to verify a transaction, Bob would query a number of different nodes for the current chain, and then search for all transactions involving the coin Alice claims belongs to her. If he receives no results from anyone indicating that her coin has been spent (and thus no longer belongs to her), then he accepts her statement that she owns the coin and accepts her transaction.
- To create anonymity in the system, Alice and Bob each creates a private key with a public counterpart. Alice and Bob could even create many different private keys if they wished to split up their holdings to prevent their coins or transactions from being linked to one another.
  - Alice then uses her public key ("Person 111920") as her identity when she signs a coin, and Bob does the same with his identity ("Person 943019") when Alice transfers ownership to him.
  - The public keys of Alice and Bob then become part of the transaction information announced to the network: the announcement in this case would essentially say "Person 111920 transferred 1 coin to Person 943019 at timestamp 201604041420201".
  - However, there is no public information that indicates that Person 111920 is Alice and Person 943019 is Bob, so the transaction is anonymous in this respect.
  - Alice can then perform a zero-knowledge proof with Bob to prove that she possesses the private key associated to her public key without having to reveal her private key. This, in combination with the signing scheme, allows her to verifiably claim ownership of her coins without revealing her identity.
  - To transfer a coin, Alice and Bob then use a blind signature procedure to allow Bob to put his public signature on the coin. The transaction is then announced and verified by the network, and will eventually be permanently incorporated into a block in the block chain.

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