Introduction to Cryptographic Protocols

- A cryptographic protocol is a procedure carried out between two parties to perform some security tasks. Typically cryptographic protocols make use of one, or more, cryptographic primitives and/or schemes.

- An example might be the transmission of a credit card number from Bob to an e-commerce web site Alice. Such a protocol might involve a digital signature scheme (so Bob knows he is talking to Alice), and a form of encryption (to ensure Bob's credit card details are not intercepted in transit).

Protocol Characteristics

- All parties involved must know the protocol and all of the steps involved.
- All parties must agree to follow the protocol
- The protocol must be unambiguous, i.e., each step must be well defined.
- The protocol must be complete.
- It should not be possible to do more or learn more than what is specified in the protocol.

- Many examples of protocols in our daily lives:

    - paying for goods at a store or online
    - a game of tennis
    - voting in an election
    - conducting a negotiating session

- Most of these rely on a face-to-face presence and social structures for their security.

- We must deal with the near certainty that some of the people we conduct business or associate with online, or in person (business associates, managers, software developers, etc.), are dishonest and in fact have criminal intent.

- Formal protocols allow us to examine ways in which the formal processes can be subverted, and take steps to mitigate these threats.

- **Dramatis Personae** (from your textbook)

    - Alice: First participant in all the protocols
    - Bob: Second participant in all the protocols
    - Carol: Participant in the three- and four-party protocols
    - Dave: Participant in the four-party protocols
    - Eve: Eavesdropper
    - Mallory: Malicious active attacker
    - Trent: Trusted arbitrator
    - Walter: Warden, he'll be guarding Alice and Bob in some
    - protocols
    - Peggy: Prover
    - Victor: Verifier

Arbitrated Protocols

- An arbitrator disinterested party (with no vested interests), known to and trusted by both Alice and Bob acts as intermediary to An arbitrator is a disinterested third party trusted facilitate and to complete a protocol.

- This intermediary has the following characteristics:

    - Disinterested: no vested interest in the protocol and no allegiance to any of the parties
    - Trusted: the parties in the protocol will accept what the arbitrator says is true (for example, lawyers, banks, or notary public (validates signatures) are often used as arbitrators when we conduct our business.

- Example: Trent (lawyer) will prevent Alice and Bob from cheating each other by being a trusted intermediary and help Bob buy a car from Alice when Bob does not trust Alice to give him the title, and Alice does not trust Bob to give her a valid check, using the following protocol:

    1. Alice give the title to the lawyer (Trent)
    2. Bob gives the check to Alice
    3. Trent waits a specified time for the check to clear, and then gives the title to Bob.
    4. If the check does not clear within the specified time, Alice presents proof of this to Trent, who returns the title to her.

- Now consider the following problems that arise in arbitrated protocols:

- There is always overhead incurred in maintaining an arbitrator – both monetary in the case of a human arbiter, or processing time in the case of a computer (network). Thus, the arbitrator becomes a bottleneck.

- The arbitrator becomes a target for anyone attempting to subvert the network, and consequently a serious level amount of damage results if the arbitrator was to be compromised.

Adjudicated protocols

- This is a variation on arbitration, where the arbitrator, referred to as an adjudicator (or judge) gets involved called in only in the case of a dispute between two parties. The premise is that both parties are going to be honest.

- In that sense the emphasis is on detecting cheating rather than preventing it.

- This protocol relies on the existence of evidence, a paper (or electronic) trail to allow the judge to make a decision on which one of the parties is dishonest.

- For example, Alice and Bob might draw up a contract agreeable to both of them, have it witnessed and sign it.

- In the case of a dispute later on, both can present their evidence, and call witnesses, before a judge.

Self-enforcing protocols

- The main principle behind this protocol is that the protocol itself guarantees fairness, prevents disputes, and detects any attempt to cheat in time to allow the protocol to be aborted without the dishonest party gaining an advantage.

- No arbitrator is necessary to complete the protocol, and no adjudicator is required to resolve disputes, thus reducing overhead, delays, and possible compromises.

- However these are very difficult to achieve in practice and every situation will require a custom protocol implementation.

Attacks on protocols

- Generally, cryptographic attacks are directed against secure systems by attacking two main areas:

- The Cryptographic algorithms used in protocols
- The Cryptographic techniques and implementations
- The protocols themselves

- **Passive attacks**

    - This type of attack is generally directed at potential weaknesses in the algorithms.
    - A third party (Eve) eavesdrops on some or all of the protocol, and attempts to obtain information they are not intended to have.
    - This activity is difficult to detect; protocols are designed to be as secure as possible against eavesdropping.
    - The methods of eavesdropping can be Eve watching only the exchange of communication between two parties, or also watching keystrokes, or reading decrypted messages.

- **Active attacks**

    - This type of an attack attempts to subvert the functionality and implementation of the cipher.
    - A third party (Mallory) attempts to alter the protocol to his own advantage - pretending to be someone else, intercepting and replaying messages, or retransmitting altered versions, or modifying the information stored in a file.
    - The main objectives of this attack include corrupting information, obtaining unauthorized information, and disrupting service.
    - Mallory might be anyone, up to and including the system administrator.

- **Cheaters**

- This attack involves the protocol itself.
- One (or more) parties involved in the protocol attempt to subvert the protocol to gain more information or to accomplish something more than is specified by the protocol.
- Passive cheaters are those who follow the protocol, but attempt to gain extra information.
- Active cheaters are those who disrupt the protocol in progress in order to cheat and accomplish their goals.

Communications Using Symmetric Cryptography

- The following interaction illustrates all of the steps involved when Alice sends an encrypted message to Bob:

    1. Alice and Bob agree on a cryptosystem
    2. Alice and Bob agree on a key
    3. Alice takes her plaintext message and encrypts it use the encryption algorithm and the key. This creates a ciphertext message
    4. Alice sends the ciphertext message to Bob
    5. Bob decrypts the ciphertext message with the same algorithm and key and reads it

- A common technique is to encrypt each individual conversation with a separate key called a session key.

- This session key is only valid for the current session.

- Generally, the following characteristics apply in Symmetric cryptosystems:

    - Security should lie in the key.
    - Keys must be distributed in secret, and are as valuable as all the information they encrypt.
    - Having a key compromised (broken, stolen, extorted, bribed), results in very serious consequences such as private messages read, and messages faked, etc.
    - N users need O (n^2) keys to communicate securely.

Key Exchange with Symmetric Cryptography

- Let us assume that Alice and Bob share a secret key with the Key Distribution Center (KDC), the (Trent in our protocol):

    1. Alice calls Trent and request a session key to communicate with Bob
    2. Trent generates a random session key. He encrypts two copies of it; one with Alice's key and one with Bob's key. He then sends both copies to Alice
    3. Alice decrypts her copy of the session key
    4. Alice sends Bob his copy of the session key
    5. Bob decrypts his copy of the session key
    6. Both Alice and Bob uses the session key to communicate with each other

- This security of this process relies entirely on Trent not being compromised.

- If Mallory corrupts Trent, the whole network is compromised

- In addition, Trent is also a bottleneck in this session.

- Symmetric Cryptography is vulnerable to the following attacks:
    - If Eve intercepts the transmission in step 4 (passive attack), a cipher-text only attack is possible since steps 1 and 2 provide all information necessary to decrypt the ciphertext.
    - An active attacker, Mallory, could break the communication paths in step 4, preventing Alice and Bob from communicating.
    - Mallory could also intercept the key in step 2 and send messages to Bob pretending to be Alice.
    - Alice or Bob can also Give the key to Eve, who then can read all messages from Bob/Alice

Key Exchange with Public-Key Cryptography

- In most practical implementations, public-key cryptography is used to secure and distribute session keys:
    1. Bob sends Alice his public key (or Alice gets it from the KDC)
    2. Alice generates a random session key, ***K***, encrypts it with Bob's public key, and sends it to Bob: $E_B[K]$
    3. Bob decrypts Alice's message using his private key and recovers the session key: $D_B(E_B[K]) = K$
    4. Both Alice and Bob use the same session key to communicate securely with each other

- This method is vulnerable to a "**Man-in-the-Middle**" attack as follows:
    - Mallory can impersonate Bob when talking to Alice and impersonate Alice when talking to Bob.
    - Alice sends Bob her public key. Mallory intercepts this key and sends Bob his own public key.
    - Bob sends Alice his public key. Mallory intercepts this key and sends Alice his own public key.
    - When Alice sends a message to Bob, encrypted in "Bob's" public key, Mallory intercepts it. Since this message is really encrypted with his own public key. He then decrypts it with his private key, re-encrypts it with Bob's public key, and sends it on to Bob.
    - When Bob sends a message to Alice, encrypted in "Alice's" public key, Mallory intercepts it. Since this message is really encrypted with his own public key. He then decrypts it with his private key, re-encrypts it with Alice's public key, and sends it on to Alice.
    - Even if the keys are stored in a database this attack will still work if Mallory can intercept those messages as well.

One-Way Functions

- One way functions are fundamental building blocks for cryptographic protocols.
- A one-way hash function is designed in such a way that it is hard to reverse the process, that is, to find a string that hashes to a given value (hence the name **one-way**.)
- The difficulty lies in the amount of computing power and resources required to break the hash would literally take millions of years to compute.
- A hash function is a function that takes a variable-length input string (called **pre-image**) and converts it to a fixed-length (generally smaller) output string (called **hash value**)
- A simplistic example is a function that takes the pre-image and returns a byte consisting of the XOR (exclusive-or) of all input bytes. This can be used as a quick check if two pre-images are the same.
- A good hash function also makes it hard to find two strings that would produce the same hash value. All modern hash algorithms produce hash values of 128 bits and higher.
- Even a slight change in an input string should cause the hash value to change drastically. Even if 1 bit is flipped in the input string, at least half of the bits in the hash value will flip as a result. This is called an **avalanche effect**.
- A one-way hash function, also known as a message digest, fingerprint or compression function, is a mathematical function which takes a variable-length input string and converts it into a fixed-length binary sequence.
- Since it is computationally not feasible to generate plaintext that would hash to a given value, a document's hash can serve as a cryptographic equivalent of the document.
- This makes a one-way hash function a central notion in public-key cryptography. When producing a digital signature for a document, we no longer need to encrypt the entire document with a sender's private key (which can be extremely slow). It is sufficient to encrypt the document's hash value instead.
- Although a one-way hash function is used mostly for generating digital signatures, it can have other practical applications as well, such as secure password storage, file identification and message authentication code (MAC).
- A one-way hash function is also known under different names: message digest, fingerprint, (cryptographic) checksum, message integrity check, manipulation detection code.
- A good one-way hash function is also **collision-free**: It is hard to generate two pre-images with the same hash value. This is also referred to as **collision resistance**.
- MD5, SHA-1 and SHA-2 (SHA-3 in future) are widely used one-way hash functions. MD5 has been shown to be very weak in terms of collision-resistance. SHA-2 is considered better in that regard.

Hash Attacks

- There are several different types of Hash attacks and these are described below in increasing order of difficulty for an attacker.

- **Collision**

    - A collision attack occurs when it is possible to find two different messages that hash to the same value.

    - A collision attack against a CA (Certificate Authority) happens at the time when the certificate issued. For example, in an MD5 attack, the attacker was able to produce a pair of colliding messages, one of which represented the contents of a benign end-entity certificate, and the other of which formed the contents of a malicious CA certificate.

    - Once the end-entity certificate was signed by the CA, the attacker reused the digital signature to produce a fraudulent CA certificate.

    - The attacker then used their CA certificate to issue fraudulent end-entity certificates for any domain. Collision attacks can be mitigated by putting entropy into the certificate, which makes it difficult for the attacker to guess the exact content of the certificate that will be signed by the CA.

    - Entropy is typically found in the certificate serial number or in the validity periods. SHA-1 is known to have weaknesses in collision resistance.

- **Second-preimage**

    - In a second-preimage attack, a second message can be found that hashes to the same value as a given message.

    - This allows the attacker to create fraudulent certificates **at any time**, not just at the time of certificate issuance. SHA-1 is currently resistant to second-preimage attacks.

- **Pre-image**

    - A pre-image attack is against the one-way property of a hash function. In a pre-image attack, a message can be determined that hashes to a given value.

    - This could allow a password attack, where the attacker can determine a password based on the hash of the password found in a database. SHA-1 is currently resistant to preimage attacks.

- Attacks against hash functions are measured against the length of time required to perform a brute-force attack, in which messages are selected at random and hashed until a collision or preimage is found.

- According to the **birthday paradox** (coming in the key length module), the time required to find a collision by brute force is approximately $2^{n/2}$, where n is the bit length of the hash.

- To find a preimage or second-preimage by brute force, approximately 2^n messages must be hashed. Thus, a hash function is considered broken if a collision can be found in less time than that needed to compute $2^{n/2}$ hashes, or if a preimage or second-preimage can be found in less time than would be needed to compute 2^n hashes.

- For common hashes the bit length is: MD5 (128 bits), SHA-1 (160 bits) and SHA-2 (224, 256, 384, or 512 bits).

- MD5 and SHA1 were designed for a number of properties, one of the strongest being **collision resistance**. This means that it should take an enormous computational effort - around **2^{64} MD5** calls or **2^{80} SHA1** calls in order to find two files that have the same hash output.

- The time required to perform a brute-force attack keeps getting shorter due to increases in available computing power (Moore's Law). Thus, increases in hash function lengths are necessary to maintain an acceptable margin of security.

- In the past, an attack threshold of 2^{64} operations was considered acceptable for most uses, but NIST (National Institute of Standards and Technology) now recommendations now set the threshold at 2^{80}, and this will soon move up to 2^{112}.

- Using the formula $2^{n/2}$, a brute-force attack against SHA-1 would require 2^{80} computations. However, security researchers have discovered an attack strategy that requires only 2^{61} computations.

- SHA1 is broken in theory. There's an attack that should produce two colliding files in about 2^{61} SHA1 calls. That's cheap enough that a large company or government agency could carry out the attack, but expensive enough that nobody has actually done it (in public anyway).

- In fact, Bruce Schneier has estimated that the cost of a performing SHA-1 collision attack will be within the range of organized crime by 2018 and for a university-level project by 2021.

- This is an important property for a number of applications, most obviously digital signatures. For signatures, a file is usually hashed and then signed. So if two files -- one innocuous and one malicious -- can be prepared with the same hash values, a signature on the innocuous one will also be valid on the malicious one.

- This is how the Flame malware, for example, posed as a legitimate windows patch (using an MD5 collision not on Flame itself, but on the certificate used to sign it).

- MD5 is completely broken for this purpose, and new collisions with chosen properties can be generated on a PC within minutes.

Digital Signatures

- The digital equivalent of a handwritten signature or stamped seal, but offering far more inherent security, a digital signature is intended to solve the problem of **tampering** and **impersonation** in **digital communications**.

- Digital signatures can provide the added assurances of evidence to origin, identity and status of an electronic document, transaction or message, as well as acknowledging informed consent by the signer.

- In most countries, digital signatures have the same legal significance as the more traditional forms of signed documents. For example, the US Government Printing Office publishes electronic versions of the budget, public and private laws, and congressional bills with digital signatures.

- Digital signatures are based on public key cryptography, also known as asymmetric cryptography. Using a public key algorithm such as RSA, one can generate two keys that are mathematically linked: one private and one public.

- To create a digital signature, signing software (such as an email program) creates a one-way hash of the electronic data to be signed. The private key is then used to encrypt the hash.

- The encrypted hash - along with other information, such as the hashing algorithm - is the digital signature. The reason for encrypting the hash instead of the entire message or document is that a hash function can convert an arbitrary input into a fixed length value, which is usually much shorter. This saves time since hashing is much faster than signing.

- A digital signature is a mechanism by which a message is authenticated i.e. proving that a message is effectively coming from a given sender, much like a signature on a paper document.

- For instance, suppose that Alice wants to digitally sign a message to Bob. To do so, she uses her private-key to encrypt the message; she then sends the message along with her public-key (typically, the public key is attached to the signed message).

- Since Alice's public-key is the only key that can decrypt that message, a successful decryption constitutes a Digital Signature Verification, meaning that there is no doubt that it is Alice's private key that encrypted the message.

Signing Documents with Symmetric Cryptosystems and an Arbitrator

- Alice wants to sign a digital message and send it to Bob, using Trent and a symmetric cryptosystem. Trent is a powerful, trusted arbitrator. He can communicate with both Alice and Bob.

- Trent shares a secret key, K_A, with Alice, and a different secret key, K_B, with Bob:

    1. Alice encrypts her message to Bob with K_A and sends it to Trent.
    2. Trent decrypts the message with K_A.
    3. Trent takes the decrypted message and a statement that he has received this message from Alice, and encrypts the whole bundle with K_B.
    4. Trent sends the encrypted bundle to Bob.
    5. Bob decrypts the bundle with K_B. He can now read both the message and Trent's certification that Alice sent it.

- Trent knows that the message is from Alice and not from some imposter because he infers it from the message's encryption. Only Trent and Alice share their secret key, only Alice could encrypt a message using it.

- Now, if Bob wants to show Carol a document signed by Alice, he cannot reveal his secret key to her, so he has to go through Trent again:

    1. Bob takes the message and Trent's statement that the message came from Alice, encrypts them with K_B, and sends them back to Trent.
    2. Trent decrypts the bundle with K_B.
    3. Trent checks his database and confirms that the original message came from Alice.
    4. Trent re-encrypts the bundle with the secret key he shares with Carol, K_C, and sends it to Carol.
    5. Carol decrypts the bundle with K_C. She can now read both the message and Trent's certification that Alice sent it.

- In theory these protocols work, but they are not very practical. For example, the intermediary software (Trent) must spend an enormous amount of time decrypting and encrypting messages, acting as the intermediary between every pair of people who want to send signed documents to one another.

- He must keep a database of messages (although this can be avoided by sending the recipient a copy of the sender's encrypted message). He is a bottleneck in any communications system, even if he's a mindless software program.

- Even more important is maintaining the intermediary. This application must be absolutely infallible. Even a single mistake in a million signatures will result in complete loss of trust and confidence (with enormous financial consequences).

- In addition, Trent has to be completely secure. If his database of secret keys ever got out or if someone managed to modify his programming, everyone's signatures would be completely useless. False documents purported to be signed years ago could appear causing chaos and collapse of public institutions.

Signing Documents with Public-Key Cryptography

- **Asymmetric Cryptography**, also known as public key cryptography, uses public and private keys to encrypt and decrypt data. These can also be used for digital signatures.

- In some algorithms, RSA is an example; either the public key or the private key can be used for encryption. Encrypting a document using a private key is essentially a secure digital signature.

- The basic protocol is simple:

    1. Alice encrypts the document with her private key, thereby signing the document.
    2. Alice sends the signed document to Bob.
    3. Bob decrypts the document with Alice's public key, thereby verifying the signature.

- This protocol is far better than the previous one. Trent is not needed to either sign or verify signatures. (He is needed to certify that Alice's public key is indeed her public key.)

- The parties do not even need Trent to resolve disputes: If Bob cannot perform step (3), then he knows the signature is not valid.

Signing Documents with Public-Key Cryptography and One-Way Hash Functions

- From a practical standpoint, public-key algorithms are usually too inefficient to sign long documents. For the purposed of efficiency, digital signature protocols are often implemented with one-way hash functions.

- Instead of signing a document, Alice signs the hash of the document. In this protocol, both the one-way hash function and the digital signature algorithm are agreed upon beforehand.

    1. Alice produces a one-way hash of a document.
    2. Alice encrypts the hash with her private key, thereby signing the document.
    3. Alice sends the document and the signed hash to Bob.
    4. Bob produces a one-way hash of the document that Alice sent. He then, using the digital signature algorithm, decrypts the signed hash with Alice's public key. If the signed hash matches the hash he generated, the signature is valid.

- This mechanism results is a dramatic increase in speed since the chances of two different documents having the same 160-bit hash are only one in 2^{160}, and anyone can safely equate a signature of the hash with a signature of the document.

- If a non-one-way hash function were used, it would be an easy matter to create multiple documents that hashed to the same value, so that anyone signing a particular document would be fooled into signing a multitude of documents.

- In addition, the signature can be kept separate from the document. Also, the recipient's storage requirements for the document and signature are much smaller.

- An archival system can use this type of protocol to verify the existence of documents without storing their contents. The central database could just store the hashes of files.

- If there is any disagreement in the future about who created a document and when, the database could resolve it by finding the hash in its files.

Digital Certificates

- A certificate is a piece of information that proves the identity of a public-key's owner. Like any government-issued ID or document, a certificate provides recognized proof of a person's (or entity) identity.

- Certificates are signed and delivered securely by a trusted third party entity called a **Certificate Authority** (CA).

- As long as Bob and Alice trust this third party, the CA, they can be assured that the keys belong to the persons they claim to be.

- A certificate contains among other things:

    1. The CA's identity
    2. The owner's identity
    3. The owner's public-key
    4. The certificate expiry date
    5. The CA's signature of that certificate

- With a certificate instead of a public-key, a recipient can now verify a few things about the issuer to make sure that the certificate is valid and belongs to the person claiming its ownership:

    1. Compare the owner's identity
    2. Verify that the certificate is still valid
    3. Verify that the certificate has been signed by a trusted CA
    4. Verify the issuer's certificate signature, hence making sure it has not been altered.

- Bob can now verify Alice's certificate and be assured that it is Alice's private-key that has been used to sign the message.

- Alice must never divulge her private-key. This is imperative in order to enforce one aspect of the non-repudiation feature associated with her digital signature.

- Certificates are signed by a CA, which means that they cannot be altered. In turn, the CA signature can be verified using that CA's certificate.

- Certificate validation can also be added to the process. When Alice encrypts a message for Bob, she uses Bob's certificate.

- Prior to using the public-key included in Bob's certificate, some additional steps are performed to validate Bob's certificate:

    1. Validity period of Bob's certificate
    2. The certificate belongs to Bob
    3. Bob's certificate has not been altered
    4. Bob's certificate has been signed by a trusted CA

Secret Splitting

- Secret Splitting, (a variation is Secret Sharing) in cryptography, is a method to split a message into several blocks. All blocks are required to retrieve the original information.

- It is mathematically impossible to obtain the original information if one of the blocks of data is not available. The information, obtained from separate blocks does not reveal any information or partial information about the original, and does not assist in any way in retrieving the original information.

- Secret Splitting is useful in situations where secret information should be in the custody of two or more individuals, without disclosing the secret information itself to the individuals.

- Consider an example where a company has designed and implemented a very innovative application that will results in a huge amount of wealth for the company. The critical portion of the application is a series of secret algorithms.

- The secret algorithms can be divulged to a small group of the most trusted engineers and developers. However, the weakness here is if one of them defects to the competition. Or, one the members of the group's computer files are compromised.

- This is an ideal situation for **secret splitting**. The overall set of algorithms is divided up into individual components. Each component by itself will not reveal much about the overall application innovation and performance.

- Thus if any employee resigns with his single component, that information is useless by itself.

- All individuals with a share of the secret information must agree upon merging all components to retrieve the original and complete application, and no single individual can obtain that information without the authorization and the aid of all other individual holders of the remaining components.

- The principle of Secret Splitting is based on **one-time pad** encryption and is very simple but effective. Only one-time pad offers the absolute security, required to create the individual shares.

- The simplest sharing scheme splits a message between two people. Here's a protocol in which Trent can split a message between Alice and Bob:

    1. Trent generates a random-bit string, **R**, the same length as the message, **M**.
    2. Trent XORs **M** with **R** to generate **S**: $S = M \oplus R$
    3. Trent gives *R* to Alice and *S* to Bob.

- To reconstruct the message, Alice and Bob have only one step to do:

    1. Alice and Bob XOR their pieces together to reconstruct the message: **S**: $M = S \oplus R$
- This technique, if done properly, is absolutely secure. Essentially, Trent is encrypting the message with a one-time pad and giving the ciphertext to one person and the pad to the other person.

- It is easy to extend this scheme to more people. To split a message among more than two people, XOR more random-bit strings into the overall package.

- In the following example, Trent divides up a message into four pieces:

    1. Trent generates three random-bit strings, **R, S, and T**, the same length as the message, **M**.
    2. Trent XORs **M** with the three strings to generate **U**:

    $$U = T \oplus S \oplus M \oplus R$$

    3. Trent gives **R** to **Alice**, **S** to Bob, **T** to Carol, and **U** to Dave.

- Alice, Bob, Carol, and Dave, working together, can reconstruct the message:

    4. Alice, Bob, Carol, and Dave get together and compute:

    $$M = R \oplus S \oplus T \oplus U$$

- This is an adjudicated protocol. Trent has absolute power and can do whatever he wants:

    - He can hand out gibberish and claim that it is a valid piece of the secret; no one will know it until they try to reconstruct the secret.
    - He can hand out a piece to Alice, Bob, Carol, and Dave, and later tell everyone that only Alice, Carol, and Dave are needed to reconstruct the secret, and then fire Bob.

- This is not necessarily a problem since this is Trent's secret to divide up in the first place.

- However, this protocol does haves a problem:

    - If any of the pieces gets lost and Trent isn't around, no one can re-build the original block.
    - If Carol goes to work for the competition and takes her piece with her, the rest of them are out of luck.

- She can't reproduce the recipe, but neither can Alice, Bob, and Dave working together. Her piece is as critical to the message as every other piece combined.

- This is true because *R*, *S*, *T*, *U*, and *M* all have the same length; seeing anyone of them gives the length of *M*. Remember, *M* isn't being split in the normal sense of the word; it is being XOR'd with random values.

Secret Sharing

- The classic example of this is the sharing of multiple keys to launch a nuclear attack where five officers are each given a key and it requires that at least three officers to stick their keys in the proper slots before the missiles can be launched.

- A more complicated scheme would be to have a scheme where a general and two colonels are authorized to launch the missile, but if the general is not available, then five colonels are required to initiate a launch.

- An even more complicated sharing scheme, called a **threshold scheme**, can do all of this and more - mathematically.

- At its simplest level, you can take any message and divide it into **n** pieces, called **shadows or shares**, such that any **m** of them can be used to reconstruct the message. More precisely, this is called an **(*m,n*)-threshold scheme**.

- With a (3,4)-threshold scheme, Trent can divide his secret application among Alice, Bob, Carol, and Dave, such that any three of them can put their shadows together and reconstruct the overall application.

- If Carol is on vacation, Alice, Bob, and Dave can do it. If Bob quits of vanishes, Alice, Carol, and Dave can do it. However, if Bob vanishes while Carol is on vacation, Alice and Dave cannot reconstruct the application by themselves.

- This idea was invented independently by Adi Shamir and George Blakley.