Massachusetts Institute of Technology
6.857: Network and Computer Security
Professors Ronald L. Rivest and Yael Tauman Kalai

Handout R8
April 10, 2020
Adrian Sy

Recitation 8: "Quiz" Review

Agenda

- One Time Pad
- Hash Functions and Applications
- Block Ciphers & Modes of Operation
- Message Authentication Codes (MACs)
- Security Scheme Definitions
  - CPA-security, CCA-security, unforgeability under CMA
- Shamir Secret Sharing (on quiz but not covered today)
- Public Key Cryptosystems
  - RSA
  - El Gamal
- Commitment Schemes (Definition, Pedersen Commitments)
- Digital Signatures
  - Hash & Sign Paradigm
- Digital Payments
- Zero Knowledge
- Miscellaneous (on quiz but not covered today, see lecture notes/videos)
  - COVID-19 Contact Tracing
  - Differential Privacy
  - Voting

1 One Time Pad (OTP)

Information theoretically secure encryption technique:

- Gen- generate random n-bit pad p
- $\text{Enc}(p, m) = m \oplus p$
- $\text{Dec}(p, c) = c \oplus p$

Pros:

- Perfectly secure/ satisfies Shannon secrecy, i.e. information about message does not increase even after seeing cipher text

  Proof idea: a ciphertext can map to any possible plaintext with the appropriate pad.

Cons:

- Requires pad/ shared secret key to be as long as the message every time
- Pad can only be used once

2 Hash Functions

Function $h : \{0,1\}^* \to \{0,1\}^d$ maps arbitrary input to a fixed length output.

2.1 Properties

- **One-wayness** - Hard to invert, i.e. given y in the output space of h, it should be hard for an adversary to find any x such that $h(x) = y$
- **Target Collision Resistance** - Hard to find a collision to a given number, i.e. given some x, it should be hard for an adversary to find any x' such that $h(x) = h(x')$
- **Collision Resistance** - Hard to find any collision, i.e. hard for an adversary to find any x, x' such that $h(x) = h(x')$
- **Pseudorandom** - Distribution looks random, i.e. the distribution of $h(x)$ for uniformly random x should look like a uniformly random sampling of the output space of h

2.2 Applications

- **Integrity Checks** - For collision resistant hash functions, the hash of an object can be used as proof of integrity of that object
- **Merkle Trees** - Uses hash of items and forming a tree structure to show integrity of the collection of items.
- **Merkle Puzzles** - Using hardness of hash inversion for encryption scheme. Hash function is typically a permutation from a limited input space. Simplified procedure below:

  1. Bob generates m puzzles each containing some unique identifier x_i and some symmetric key y_i, e.g. $h(x_i|y_i)$.
  2. Alice randomly picks a puzzle and uses brute-force methods in $O(n)$ time to invert the hash function and extract x_j and y_j, x_j is then sent back to Bob in the clear.
  3. Bob finds the y_j corresponding to x_j and they communicate using y_j as a shared secret key.

  Alice and Bob's runtime is $O(m + n)$ while an eavesdropper would need $O(mn)$.

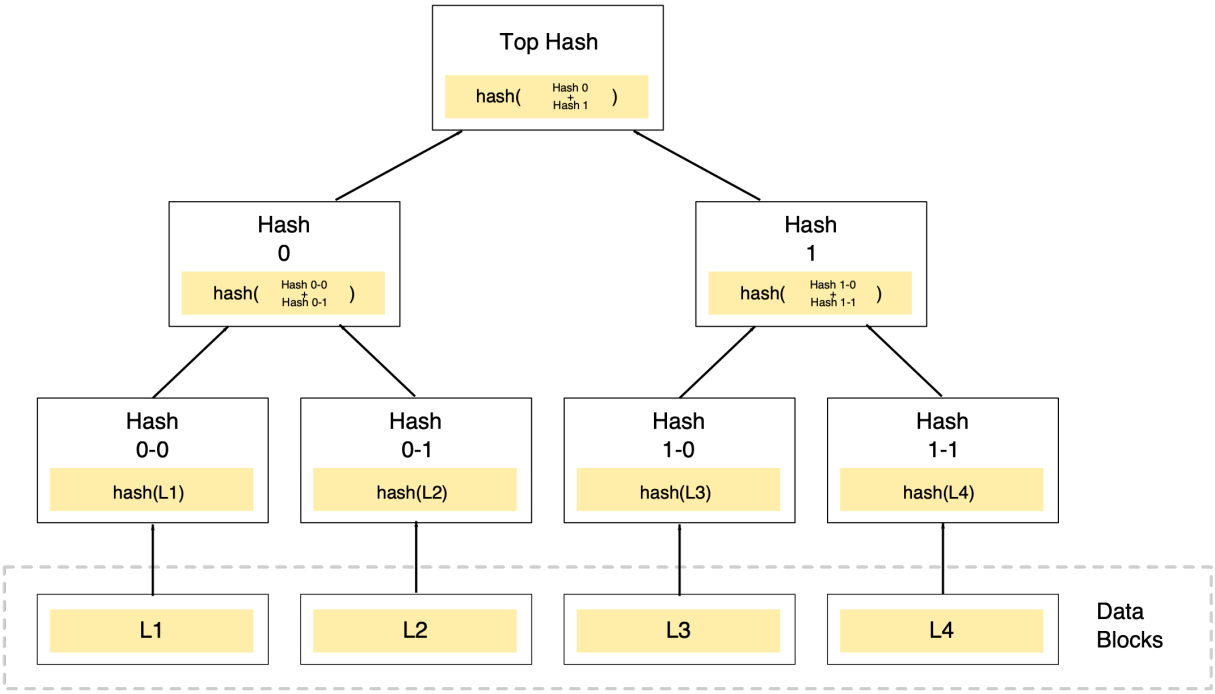

Figure 1: Diagram of Merkle Tree.[1]

3 Block Ciphers & Modes of Operation

3.1 Block Cipher

Encryption procedure that takes in fixed-length input and return fixed-length output. Easier to analyze than arbitrary length input. Some examples are DES and AES. Allow deterministic encryption and decryption using a key.

Ideal block ciphers are abstractions for block cipher security, they are effectively indistinguishable from a random permutation function.

3.2 Modes of Operation

Modes of operation are ways of chaining block ciphers together to handle arbitrary length input. The input is typically padded to make the input length a multiple of the block cipher input size. Even if the block cipher is an ideal block cipher, a bad choice of mode of operation can result in insecure encryption schemes.

3.2.1 Examples

1. **Electronic Code Book (ECB)** - Use the same key for every block cipher and just encrypt every message block. Not good because information about whether or not two blocks are the same is revealed. Also deterministic so not resistant to CPA attacks. Pro is that it is parallelizable.

[1]Source: By Azaghal - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=18157888

2. **Cipher Block Chaining (CBC)** - Have a random initialization vector (IV) be c_0, generate cipher blocks $c_i = \text{Enc}_k(c_{i-1} \oplus m_{i-1})$ where $m_0, \ldots, m_n$ are the plaintext message blocks.

3. **Cipher Feedback (CFB)** - Similar to CBC except the XOR comes after the encryption, i.e. $c_i = \text{Enc}_k(c_{i-1}) \oplus m_{i-1}$.

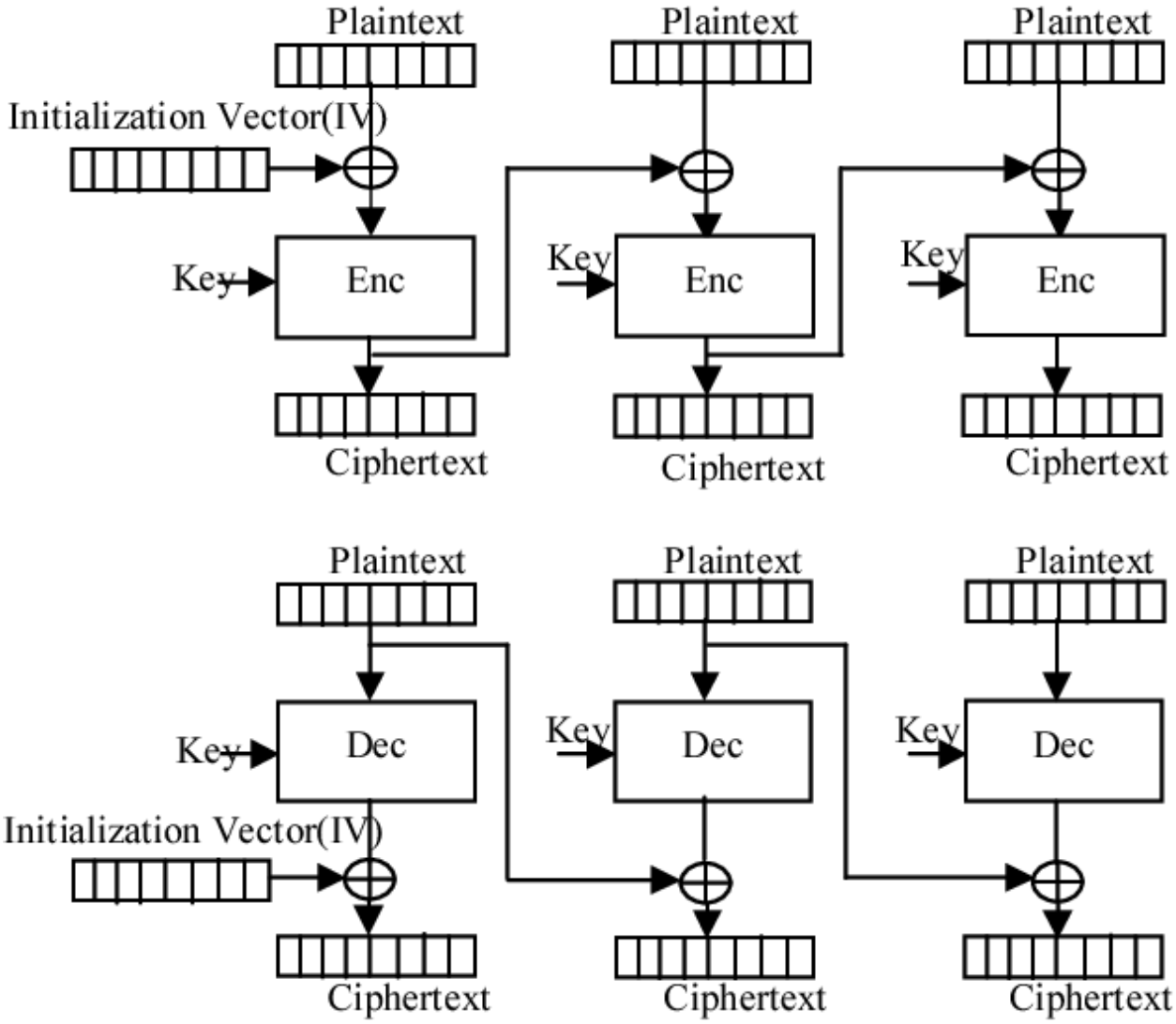

Figure 2: CBC encryption and decryption[2]

[2]Source: https://www.semanticscholar.org/paper/Design-and-analysis-of-AES-CBC-mode-for-high-Vaidehi-Rabi/632331ec119317d93837bd613a7d0a6c8d918490/figure/1

4 Message Authentication Codes (MACs)

Message Authentication Codes provide integrity to a message, i.e. makes it detectable when a message has been altered.

A MAC scheme is correct if for a validly generated MAC is always verified by the verification algorithm, i.e. $\text{Ver}(k, m, \text{MAC}(k, m))$ is always true.

A MAC scheme is sound if the probability of an invalid MAC being verified as true is negligible.

5 Security Scheme Definitions

5.1 Indistinguishability under Chosen Plaintext Attack (IND-CPA)

IND-CPA (or IND-CPA-2 for adaptive case) security means that any adversary $\mathcal{A}$ cannot efficiently win in the following game.

1. The challenger generates a key k and gives $\mathcal{A}$ access to encryption oracle Enc_k.
2. As many times as needed, $\mathcal{A}$ can generate a message m and see the ciphertext $Enc_k(m)$.
3. $\mathcal{A}$ chooses two messages m_0 and m_1 and sends them to the challenger.
4. The challenger returns ciphertext c which is either $Enc_k(m_b)$ for $b \in \{0, 1\}$.
5. (Adaptive case) $\mathcal{A}$ can continue to query the encryption oracle.
6. $\mathcal{A}$ wins if they can guess b correctly.

5.2 Existential Unforgeability under Chosen Message Attack (CMA)

Existential unforgeability under CMA is a security definition for MAC schemes (and corresponding public key variant Digital Signatures) such that an adversary cannot efficiently win in the following game.

1. The challenger generates a key k and gives $\mathcal{A}$ access to MAC oracle MAC_k.
2. As many times as needed, $\mathcal{A}$ can generate a message m and see the MAC $M = MAC_k(m)$.
3. $\mathcal{A}$ wins if they can generate a **new** message m' and MAC M' such that $Ver_k(m', M') = 1$.

5.3 Indistinguishability under Chosen Ciphertext Attack (IND-CCA)

For symmetric encryption schemes, a Chosen Ciphertext Attack (CCA) is when the adversary has access to both an encryption and decryption oracle and can use this to break the encryption scheme. The game corresponding to this type of security is as follows:

1. The challenger generates a key k and gives $\mathcal{A}$ access to encryption oracle Enc_k.
2. As many times as needed, $\mathcal{A}$ can generate a message m and see the ciphertext $Enc_k(m)$.
3. Similarly, $\mathcal{A}$ can generate a ciphertext c and see the decrypted plaintext $m = Dec_k(c)$ if valid.
4. $\mathcal{A}$ chooses two messages m_0 and m_1 and sends them to the challenger.
5. The challenger returns ciphertext c which is either $Enc_k(m_b)$ for $b \in \{0, 1\}$.
6. $\mathcal{A}$ can continue to query the encryption and decryption oracles (but cannot query for decryption of c).
7. $\mathcal{A}$ wins if they can guess b correctly.

6 Public Key Cryptosystems

A public key encryption scheme is defined by three algorithms. Genis the key generation algorithm where a public key pkand a secret key skare generated. Encis the encryption algorithm which takes in pkand the message m to output a ciphertext c. Decis the decryption algorithm which takes in skand ciphertext ţo output m.

By necessity, any PK encryption scheme must be randomized to be secure under CPA, otherwise the adversary can just replay requests and match values.

6.1 Textbook RSA

Key Generation - Public Key = (n, e) where $n = p * q$ (p and q are large primes) and e is a number that is coprime to $\varphi(n)$[3].

Private Key = d where $d = e^{-1} \mod \varphi(n)$

Encryption - $y = x^e \mod n$

Decryption - $x = y^d \mod n$

This scheme relies on the RSA assumption (factoring is hard). As is, it is not secure at all because it is deterministic. See recitation notes (R5) for how to improve this scheme.

6.2 El Gamal

Key Generation - Public Key = (G, q, g, h) where G is a group of order q with generator g and $h = g^x$ for some $x \in \mathbb{Z}_q^*$

Secret Key = x

Encryption - choose random $y \in \mathbb{Z}_q^*$ compute $s = h^y$ return two part ciphertext $(c_1, c_2) = (g^y, m \cdot s)$

Decryption - compute $s = c_1^x$ (evaluates to the same s in the encryption step) compute s^{-1} in the group G return $m = c_2 \cdot s^{-1}$

Relies on the Diffie-Hellman assumption (hard to determine g^{ab} given (g, g^a, g^b)). Computational Diffie-Hellman (CDH) assumption is that g^{ab} is hard to compute. Decisional Diffie-Hellman (DDH) assumption is that g^{ab} is hard to distinguish from g^u for random u.

IND-CPA requires DDH assumption to hold in G. Note that DDH does not hold in $\mathbb{Z}_p^*$.

Not IND-CCA as is because scheme is malleable.

7 Commitment Schemes

Cryptographic equivalent of a safe. Contains some value that cannot be easily identified but can be revealed easily with some code/ combination.

Commit phase - sender commits to a message m and sends commitment $\text{Com}(m)$ to the receiver

Reveal phase - sender sends some key r and receiver is able to open, i.e. $\text{Open}(r, \text{Com}(m)) = m$ or reject if invalid.

Security of a commitment scheme is defined by its hiding and binding properties. A commitment scheme is hiding if it is hard for the adversary (receiver) to get information about m from $\text{Com}(m)$. A commitment scheme binding if it is hard for the adversary (sender) to use a receive key r' such that $\text{Open}(r', \text{Com}(m)) = m'$ and $m \neq m'$. Similar to encryption security, hiding and binding can be perfect (similar to OTP) or computational (relies on hardness of some assumption, e.g. RSA).

[3] $\varphi(n)$ is the number of integers in $\mathbb{Z}_n$ that are relatively prime to n, in this case $(p-1)(q-1)$.

7.1 Pedersen Commitment

The Pedersen Commitment scheme uses a key (g, h), where g, h are elements of group G of prime order p. To commit to a message $m \in \mathbb{Z}_p$, choose a random element in $r \in \mathbb{Z}_p$ and send $c = g^m \cdot h^r$.

Perfectly hiding because any m can produce any c with the appropriate choice of r, i.e. m is the discrete log of $c \cdot h^{-r}$. Only computationally binding because it depends on the hardness of discrete log.

8 Digital Signatures

Cryptographic primitive in a public key setting for verifying authenticity / integrity. Analogous to MAC schemes in the symmetric key setting.

Defined by the following three algorithms:

- $\text{Gen}(1^k) = (\text{sk}, \text{pk})$ - generates a secret key and public key pair
- $\text{Sign}(m, \text{sk}) = t$ - generates a signature (also referred to as tag t) for message m
- $\text{Ver}(m, t, \text{pk})$ - verifies if the tag t is a valid signature for message m

8.1 Applications of Digital Signatures

The applications of Digital Signatures fall under three major categories:

1. Authentication - the tag t for message m proves that message m was created by the party with public key pk and not anyone else (because the signature is unforgeable).
2. Integrity - since the tag t only verifies for message m, then modifying m is impossible without creating a corresponding t' to the modified m'
3. Non-repudiation - the party who generated pk cannot deny having made the message m with tag t since t could not have been generated by anyone else.

8.2 Hash and Sign

Signing algorithm can be time consuming to do especially for arbitrary length messages. As mentioned above, the hash of a message is a good integrity check for the message so instead of signing the whole message, one can instead sign the hash. Other interesting properties noted in Pset 4.

9 Digital Payments

9.1 Blockchain Protocol

From a high-level perspective, the protocol proceeds as follows:

1. The blockchain starts with a genesis block that is broadcasted to everyone on the network.
2. Whenever a transaction occurs, the one giving bitcoin creates a transaction record, signs it and broadcasts it to the network.
3. Miners can generate blocks by accumulating several valid transactions that are not yet on the chain and showing proof of work. The block is then broadcasted.
4. Each user on the network adds the first block that gets broadcasted and assuming enough parties are honest, consensus will be reached.

9.2 Security

9.2.1 Digital Signatures

Each transaction has to be signed so that malicious miners cannot forge transactions to give themselves more cryptocurrency. Since the transaction records are unforgeable, the worst a malicious miner can do is to not include the transaction into a block. However, the miners are incentivized to do so because transactions typically include a small portion of currency that the miner can give themselves without breaking the signature validity.

9.2.2 Sybil Attacks

Sybil attacks refer to attacks which can be exploited by making multiple accounts (since the only identifier is a public key). For example, if mining a block is easy, a miner can simply create multiple accounts to mine simultaneously. The blockchain avoids this by using proof of work to limit the each user to the amount of computational power available to them.

9.2.3 Hash Functions

Hash functions are used in two places: to identify the previous block and to provide proof of work.

- To connect to the existing blockchain, each new block must contain the hash of the block preceding it. This prevents a malicious entity from making a block in advance and just appending it to the current chain, this entity must also be faster than all the other miners in the system. This requires the hash function to be collision resistant.

- To provide proof of work, the miner must find a nonce such that when the nonce value is combined with the other data and hashed, the resulting hash value is part of a limited subset of the possible output values, e.g. has k leading 0 bits. This requires the hash function to be pseudorandom.

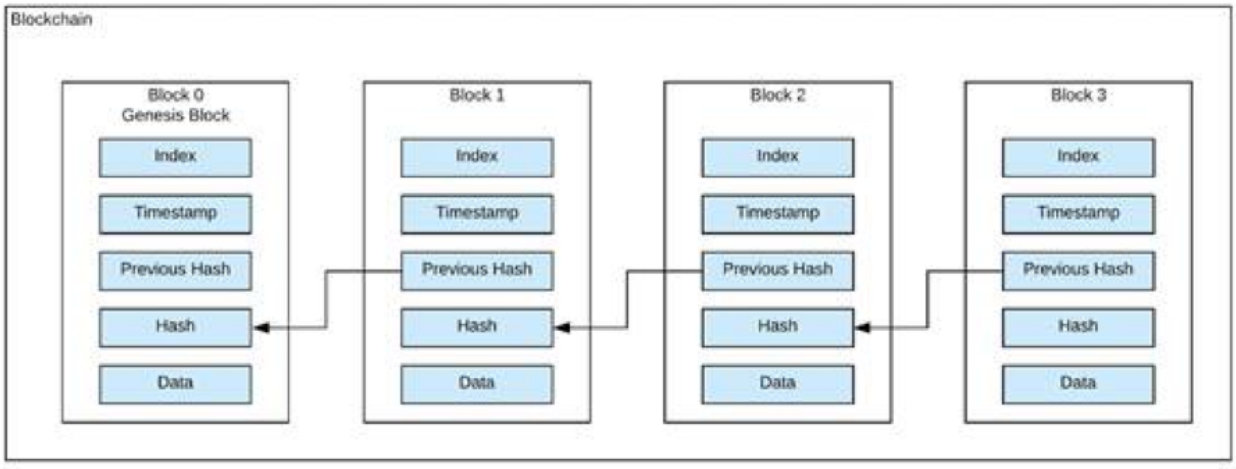

Figure 3: Blockchain Diagram.[4]

[4]Source: https://www.spheregen.com/wp-content/uploads/2019/04/blockchain.png

10 Zero-Knowledge Proofs

Interactive protocols for a prover P to show a verifier V that some item x is in some language/ set of items L. The examples below will talk about graph homomorphism, i.e. two graphs are homomorphic if there is some reordering of vertices that transforms one to another.

- **Completeness** - An honest verifier accepts a proof if the statement is true, e.g. the graphs are homomorphic
- **Soundness** - Verifier rejects the proof if the statement is incorrect with non-negligible probability (can be compounded over multiple interactions)

10.1 Graph Homomorphism Example

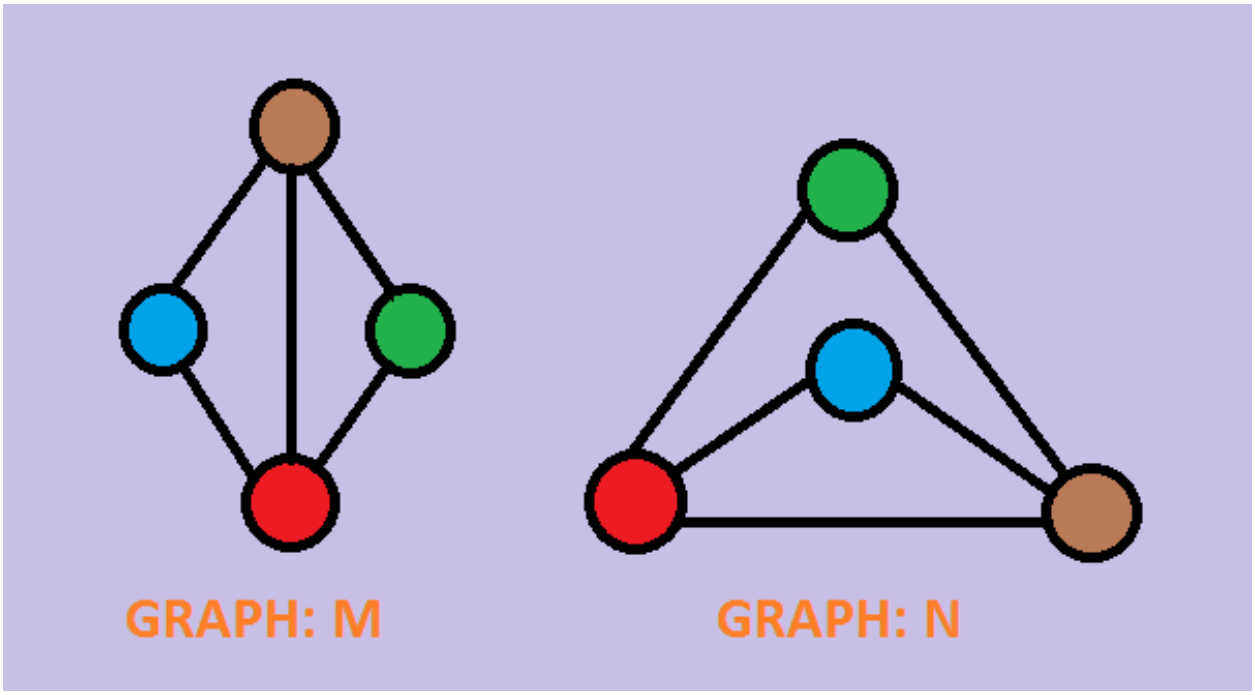

Figure 4: Homomorphic Graphs.[5]

Consider the following scheme:

1. The prover picks one of the two graphs and generates a third graph homomorphic to that graph (random permutation of vertices), by transitivity of homomorphism, all three are homomorphic (transformation to the other graph is a composition of the permutations).
2. For each of the original graphs, P commits to the permutation π that transforms it to the third graph, and commits to the third graph.
3. The verifier picks a (random) bit and sends this to the prover.
4. The prover reveals the third graph and the transformation to the corresponding graph chosen by the verifier. The verifier checks if the transformation is valid.

[5]Source: http://computingforbeginners.blogspot.com/2013/01/homomorphism-vs-isomorphism.html

- **Completeness** - The prover can always produce valid transformations if they know the homomorphism between the original graphs
- **Soundness** - If the graphs are not homomorphic, one of the transformations must be wrong, so with probability half (if step 3 is random) the verifier can reject the proof.

10.2 Simulation

To prove a scheme is zero knowledge, a simulator can without knowing any thing more than the verifier generate a transcript which is indistinguishable from the actual interactions.

For graph homomorphism, a simulator can first generate the verifier chosen bit in step 3, then produce only correct commits and transformations for the chosen graph, the other commitment can be wrong but by the hiding property of commitments this is not distinguishable from the actual transcript.