CMPSCI 677 Operating Systems **Spring 2023**

Lecture 26: May 15

Lecturer: Prashant Shenoy Scribe: ***Snigdha, Daniel Saunders, Arkin Dharawat, and Will Lee***

26.1 Distributed Systems Security

Three approaches to protect a system:

- Protect against invalid operations on your data.
- Protect against unauthorized invocations of your code.
- Protect against unauthorized users.

Security is more matter in a distributed system because the system is now accessible on a network. Other users can access it on top of authorized users.

26.1.1 Security and Privacy

- They are related but different concepts.
- Unauthorized access is a security issue. For example, somebody gets access to your machine or server and may cause sensitive data(e.g., credit card info personal info) to be stolen.
- Privacy violations can occur without any security violation. Data collected from anonymous browsing history can be de-anonymized. Mobile applications collect location trace for advertising. Learn internal details of deep neural network for simple request patterns

26.1.2 Authentication

Idea is to prove to other party who you claim to be. Question is how do I know if the remote party is really who they claim they are.

26.1.2.1 Authentication Protocol (Ap)

- Ap 1.0: Simplest approach - one party/process is trying to prove to another their identity by sending a message but the problem is an intruder could also send such a message
- Ap 2.0: Another is to use IP address check and see if the party is coming from that known IP. Problem is IP spoofing where an attacker could insert a fake IP address into a packet and pretend to be coming the approved IP address
- Ap 3.0: Use usernames and passwords. Problem is attacker can look at the traffic and intercept the password through packet sniffing

- Ap 3.1: Instead of plain text, use encryption such as symmetric key.It uses a function to encrypt the message and without the right key, attacker cannot retrieve the message back.Receiver use the same key to decrypt the message. Problem is the re-play/playback attack. An attacker could intercept the encrypted message and hand it off to claim as the real sender. The receiving party could be fooled into decrypting the message to the attacker.

26.1.3 Authentication using Nonces

A nonce is an "once-in-a-lifetime-only" integer generated for a session and will never be used again.

- Ap 4.0: The nonce is used by the receiver to challenge the sender to prove their identity. Receiver B will send the nonce to the sender. Sender will use a secret key to encrypt the nonce and send it back. When the receiver gets the encrypted message and if the decrypted message is the nonce, then the sender identify if verified.

Problem is it better be once in a lifetime so you don't pick it again. Make sure nonce is large enough in terms of bits.

Question: Can you add a time stamp or the time stamp could be your nonce?
Answer: You can enhance your protocol and add some additional features for it to reuse but this is a simple protocol that doesn't depend on anything more.

26.1.4 Authentication using public keys

AP 4.0 uses symmetric keys for authenticantion. **Question**: can we use public keys? *symmetry*: DA(EA(n)) = EA(DA(n)).

AP 5.0:
Uses public key (EA) and private keys (DA)

```
    A to B: msg = "I am A"
    B to A: once in a lifetime value $n$
    A to B: msg = DA($n$)
    B computes: If EA(DA($n$)) = $n$
        then A is verified
        else A is fradulent
```

Problem: An attacker can intercept messages between sender and receiver and assume the identity of the sender. If the attacker can intercept and send its own public key when the receiver asks for it, the receiver would be tricked into decrypting the message using the attacker's public key. This is a problem of key distribution and without a secure way to distributed keys, public key encryption would not work.

26.1.5 Man-in-the-middle attack

Trudy impersonates as Alice to Bob and as Bob to Alice.

Alice		Trudy		Bob
	"I am A"		"I am A"	
			nonce n	
			DT(n)	
			send me ET	
	nonce n			
	DA(n)			
	send me EA			
	EA			

Bob sends data using ET, and Trudy decrypts and forwards it using EA (Trudy *transparently* intercepts every message).

26.1.6 Digital signatures using public keys

Goals of digital signatures:

- Sender cannot repudiate message never sent ("I never sent that").
- Receiver cannot fake a received message.

Suppose A wants B to "sign" a message M:

B can first encrypt a message with its private key - this is like signing a message and send the encrypted/signed message to A.A can then use B's public key to decrypt the message and see that it was indeed signed by B

B send DB(M) to A
A computes if EB(DM(A)) = M
        then B has signed M

But it's not efficiency if a key is used to encrypt a large document.

Question: Can B plausibly deny having sent M?

26.1.7 Message digests

Encrypting and decrypting entire messages using digital signatures is computationally expensive. Routers routinely exchange data, which do not need encryption, but do require authentication and to verify that data hasn't changed.

A message digest is a compact summary/representational of the original message, like a checksum using a hash function. A hash function H converts a variable length string to a fixed length hash value. The user will then digitally sign $H(M)$, and send both M and $DA(H(M))$ which is the signature. The receiver can verify by taking a hash of the message and then take the sender's public key, extract the message, and compare them. This can verify who sent the message and that if the message has been altered.

Important property of H: We want to minimize any hash collision. In other words, we want H to have low collision probability. Given a digest x, it is infeasible to find a message y for which $H(y) = x$. Also, it is infeasible to find any two messages such that $H(x) = H(y)$ (hash collision). If the hash function H satisfies this property, then this scheme is much more efficient than digital signatures with public keys: we need only encrypt the fixed-length hash value $H(M)$, typically much shorter than M.

Question: Is it easy to Brute Force?
Answer: What is the property you want from the hash function that can cause to have issues? For example if I give you a hash of a message. The first thing is they should have the same property. Somebody can take your public key and can extract the hash and decrypt the message. To verify that you signed it you need to take the original message and actually hash it again and compare the two hashes and if they match it means that you have signed it. Hashing function is public so it can be used by anyone. Issues with it that if we can construct some other message that gives the same hash then I can claim that you signed something else. Hash functions can produce collisions I.e. they can produce same hash for different messages. So if you signed a message that generated a hash H(M) that's some integer. If I can brute force construct some other arbitrary document that gives the same hash I can claim that you signed the other document. Good hash functions will not allow you to do that.

26.1.8 Hash functions: MD5

MD5 is no longer secure to use anymore

MD5 takes an arbitrarily-sized objects, splits it into smaller chunks, and hash each of the chunks. This method is recursed until we have a fixed-sized hash value.

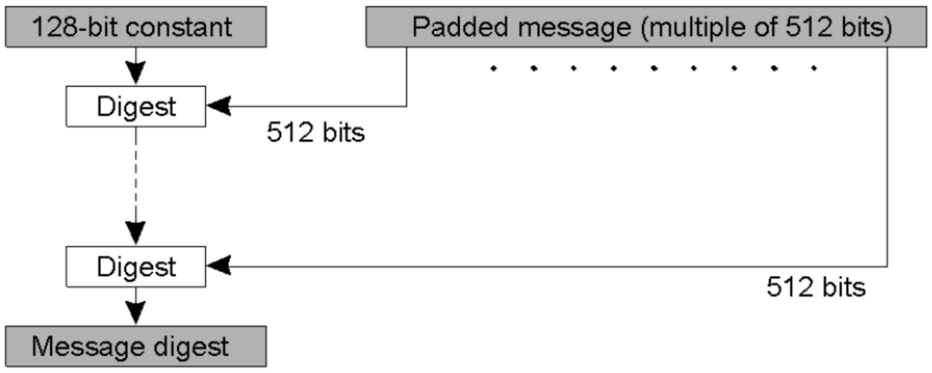

Figure 26.1: The structure of MD5.

26.1.9 Hash functions: SHA

MD5 is not secure anymore. Secure Hash Algorithms (SHA) hash functions:

- SHA-1: 160-bit function that resembles MD5.
- SHA-2: family of two hash functions (SHA-256 and SHA-512).
- Developed by NIST and NSA.

Essentially, the larger the hash, the more secure but also more expensive to compute.Additionally, probabilities of collision and being attack are lower too

Let's say you have a hashed value h, and you want to know the original message m such that $H(m) = h$. How can we find out m? We could carry out a dictionary attack, in which we try all m' in some "dictionary" and compute $H(m')$. If any such m' produces $H(m') = h$, then $m' = m$. Clearly, the longer the message, the more time required by this brute-force attack.

26.1.10 Symmetric key exchange: trusted server

Problem: How do distributed entities agree on a key? How do you exchange keys securely?

This is the problem of key distribution. If keys are comprised, than anyone may use your key to decrypt your messages. We need a secure method of key exchange; it must be just as strong as your encryption algorithm itself.

Assume: Each entity has its own single key, which only it and a trusted server know.

Server:

- Will generate a one-time session key (symmetric) that A and B use to encrypt all communication.
- Will use A and B's single keys to communicate session key to A and B.

26.1.11 Key Exchange: Key Distribution Center

Trusted third party as the key distribution center. Alice sends a message to the KDC saying "I'm Alice, and I want to communicate with Bob; please generate a key". The KDC generates a random session key, encrypts it with Alice's public key, and sends it back to Alice. The KDC will send the same key to Bob, encryped with Bob's public key.

Public key crytography is only used to key distribution. Once keys are distributed, the sender and receiver would use the same key.

Question: Can an attacker intercept the generated key? An attacker can intercept these messages but would not be able to decrypt the messages to get the generated key.

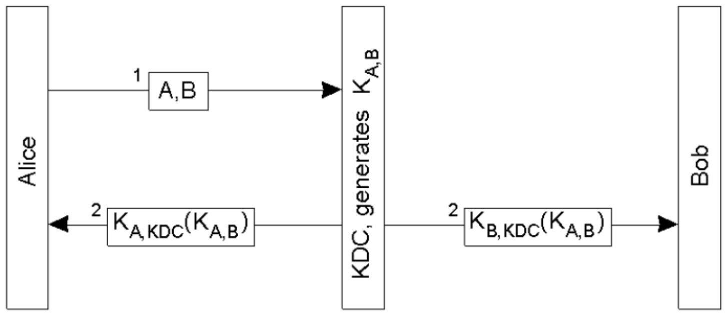

Figure 26.2: The principles of using a KDC.

26.1.12 Authentication using a key distribution center

Pictured below: Same technique as before, but the KDC does not actually send a message to Bob. Messages are sent from Alice to Bob.

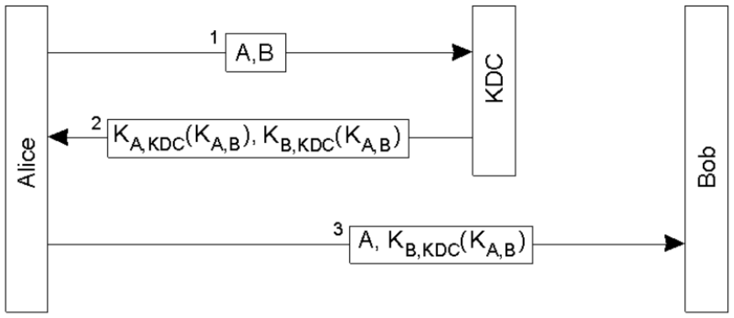

Figure 26.3: Using a ticket and letting Alice set up a connection to Bob.

26.1.13 Public Key Exchange

There is no KDC in this protocol; public and private keys are used only. This protocol assumes we have a secure way of getting someone's public key before communicating with them. Alice sends a nonce to Bob, which is encrypted by Alice with Bob's public key. Bob decrypts it, sends it back, sends a new nonce and session key back (all of this is encrypted with Alice's public key). Alice decrypts this message, and sends back Bob's generated nonce for validation purposes.

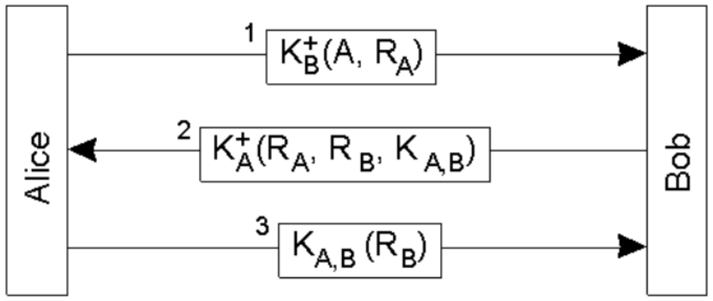

Figure 26.4: Mutual authentication in a public-key cryptosystem.

Public key retrieval is subject to man-in-the-middle attacks if public keys aren't shared securely. In other words, attackers can get hold of the actual public keys and replace them with theirs. We can use a trusted third party in order to distributed public keys securely. This third party will issue "certificates," public keys of servers which are signed by the trusted third party. Hence, when someone get your public key, it would come with a certificate with your name from a trusted certification authority and signed with its private key.

This is also how all web browsers communicate with the server using HTTPS. It would receive a certificate, verify it. Browsers have been pre-loaded with certifciate authority's public keys and could decrypt certificate for validation.

All servers have the trusted server's encryption key. Suppose that A wants to send B a message using B's "public" key; A can accomplish this by using certificates from the trusted third party. Certificates may be revoked.

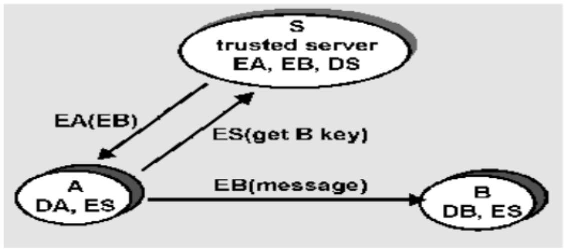

Figure 26.5: Public key exchange: trusted server.

26.1.14 Security in Enterprises

- In the multi-layered approach to security, Security functionality is spread across multiple components.
- Firewalls
- Deep packet inspection
- Virus and email scanning
- VLANS
- Network radius servers
- Securing WiFi
- VPNs
- Securing services using SSL, certificates, kerberos

Distributed systems security or network security are very complex problems. There are many layers of security protections that are deployed, and things get complicated fast. Companies typically get hacked because there is a small issue with one or more of the security items listed above.

26.1.15 Security in internet services

How can we secure internet services?

- Websites
  - Ensure all connections between browser and server are encrypted.
  - Authenticate user (username and password checks).
  - Use techniques like CAPTCHAs to prevent scripts from launching attacks.
- Challenge-response authentication

  - Push a one-time key to a user's phone to enter for further authentication.

- Two-factor authentication
  - Password and mobile phone (gmail).
- One-time passwords
- Online merchant payments: paypal, amazon payments, google checkouts

26.1.16 Firewalls

A firewall is a machine that sits on the boundary of the internal (e.g., home WiFi) and external networks (e.g., the Internet). All packets and traffic between the internal and external networks must pass through the firewall, which has a pre-configured set of packet-filtering rules. If a packet matches one of the firewall's rules, it is allowed to pass; otherwise, it is dropped.

Allow rules let the packet pass through. Deny rules block the packet from passing through. To block some traffic, one can set rules to deny packet from certain IP address.Rules can also set for outgoing traffic to deny access of certain web sites

- Firewall and Packet Filtering Firewalls: rules are used to either keep or drop packets. They don't look at contents. Only the headers of the packets.
- Application Gateways: high level firewalls look inside the packets (aka deep packet inspection) by intercepting packets for inspection to find potential virus signatures

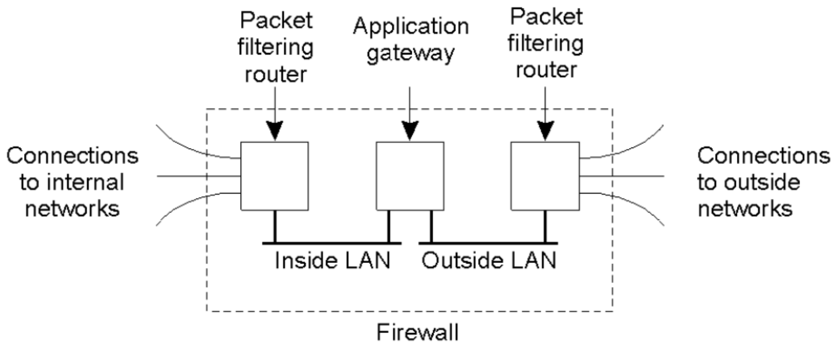

Figure 26.6: A common implementation of a firewall.

26.1.17 Secure email

- Requirements
  - Secrecy: No one else should be able to read your email other than the one intended
  - Sender authentication: Receipent should be able to verify who send the email
  - Message integrity: No one should be able to alter the content of the email

  - Receiver authentication: Only the appropriate receiver should be able to read the email

- Secrecy
  - Can use public keys to encrypt messages (this is inefficient for long messages because public keys are very long).
  - Use shorter, message-specific symmetric keys
    - Alice generates symmetric key K
    - Encrypt message M with K
    - Encrypt K with E_B
    - Send $K(M), E_B(K)$
    - Bob decrypts using his private key, gets K, and decrypts $K(M)$
- Authentication and Integrity (without secrecy)
  - Alice applies has function H to M (H can be MD5 or SHA)
  - Creates a digital signature $D_A(H(M))$
  - Send $M, D_A(H(M))$ to Bob
- Putting it all together
  - Compute $H(M)$, sign the hash $D_A(H(M))$ with the public key
  - M' = Concat the message and the hash: $\{M, D_A(H(M))\}$
  - Generate symmetric key K, compute $K(M')$
  - Encrypt K using public key as $E_B(K)$
  - Send $K(M'), E_B(K)$
- Used in PGP (pretty good privacy)

26.1.18 Secure sockets layer (SSL)

SSL (developed by NetScape) provides data encryption and authentication between web servers and clients. It lies above the transport layer. It is used for internet commerce, secure mail access (IMAP), and more. Features include: SLL server authentication, encrypted SSL sessions, and SSL client authentication.

Take a regular socket and add an encryption library on top. Send a message on the socket and the library will encrypt it. Socket on the other hand would then decrypt the message using the library.

It uses the HTTPS protocol instead of HTTP. The browsers sends the first message to the server saying it can support some version of SSL, and the server responds with its supported version of SSL, as well as a certificate (server's RSA public key encrypted by a trusted third party's (certification authority) private key). The browser generates a session key K, and encrypts the key with the public key E_S from the certificate authority. The browser sends "future messages will be encrypted" and $K(M)$ to the server, and the server responds with the same. The SSL session then begins.

Question: Are the keys store in the cookies of the clients? No. It's a session key and keep only in memory not on disk.Every time is connected, it will re-do for every socket.

26.2 Electronic payment systems

Payment systems based on direct payments between customer and merchant. Not the same as cryptocurrency, instead this is focused on using an electronic version of money instead of paper.

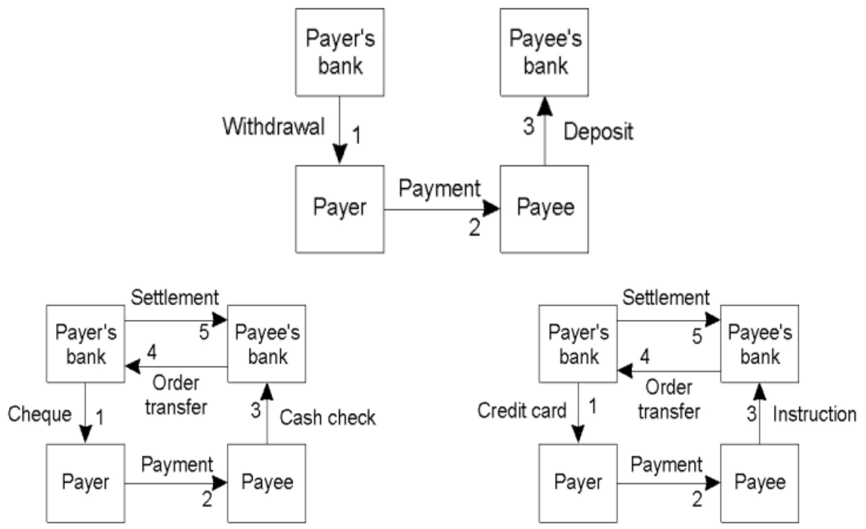

Figure 26.7: (a) Paying in cash; (b) Using a check; (c) Using a credit card.

26.2.1 E-cash

Paying in cash is inherently anonymous, Electronic cash aims to preserve anonymity of payments done similar to traditional cash.
This works on the principle of anonymous electronic cash using blind signatures. Users will first generate a "coin" which will then be "blinded" (hide the sequence number of currency to the bank; this prevents tracking). This is sent to the bank to "sign" it. Afterwards the signed coin is unblinded, and subsequently given to a receiver as payment. The receiver can then check for validity with the bank, this involves checking for the signature and that the "coin" wasn't already spent. This prevent users from spending the same "coins" more than once.

Question: Would the bank need to authorize the pay? No since the payer withdraws the money and then generates the coin. There has to be an account with the bank but the bank only signs the the withdrawn money. This way the bank knows you withdrew some money but when the money is spent they can't know if its yours.

26.2.2 Bitcoin

Cryptocurrencies are a form of digital currency which act as P2P electronic cash which are decentralized. Most popular one is Bitcoin, made by Satoshi Nakamoto based on Open source crypto protocol (proof of work etc.)
The currency in Bitcoin is generated by the participants (servers) in the network, so a peer-to-peer protocol is needed. The peers are responsible for validating transactions and performing other work for the network.

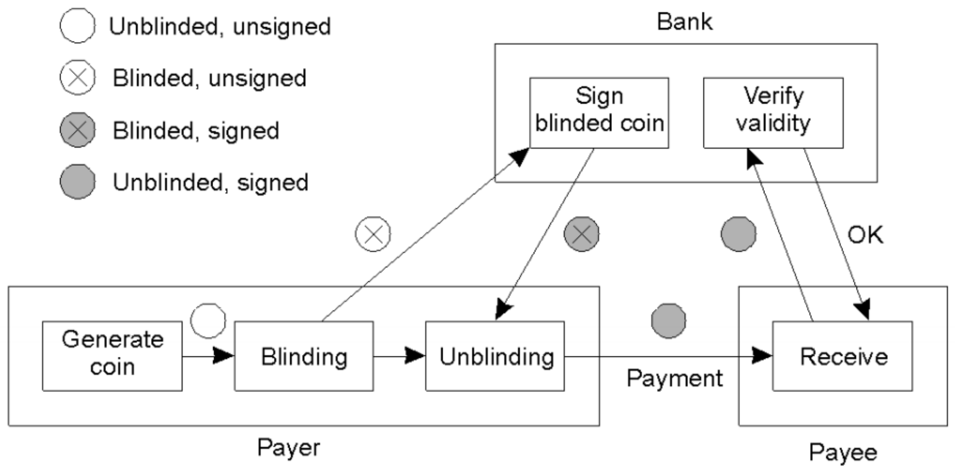

Figure 26.8: Illustration of E-cash.

All the transaction on the network are logged in a database called a Blochchain. The reward for the peers in participation is the new coins that are generated. The work to be done increases exponentially with each new coin generated, this is why from an environmental perspective this principle is not sound. Bitcoin uses digital signatures to pay ”public keys” (receiver of the payment) and all these transactions are recorded on the blockchain.
User hold bitcoins in their digital wallets. If they want to pay someone they take some bitcoin (with a unique transaction number) and transfer it's ownership to the receiver. This transfer is recorded on a public record, without putting identity of payer/receiver on the ledger since the public keys are anonymous. Anonymity comes from this, that even though the owners are not known the transaction itself is public.

26.2.3 Blockchain: Distributed Ledger

Blockchain is a public distributed database which records every transactions that goes through the system. The transactions are not recorded based on identity but on basis of public keys, but every transaction is visible to all peers in the network. Thus it is acts as a public distributed ledger. It forms the basis of cryptocurrencies but can be applied in areas beyond cryptocurrencies too, e.g: stock register/trading, land purchases, smart contracts etc.
Any transactions made is signed with a private key and added into the ledger, and this makes it public. Every block consists of multiple transactions and once committed the transactions in the block are finalized. The blockchain is massively duplicated and shared using the P2P file transfer protocols. Special nodes called “miners” that append blocks to the global transaction record. Every time a new block is generated one of the peers appends it to the network and receives a reward, in case of bitcoin the “miners” receive bitcoin. All nodes of the network perform validation and clearing of transactions. Miner nodes perform “settlement” using a distributed consensus protocol.

Question: Did the concept of a blockchain come up with bitcoin? Blockchain was a separate concept, as a distributed decentralized database. Bitcoin made it practical and incentived people to actually participate in this decentralized network that implemented a distributed ledger.

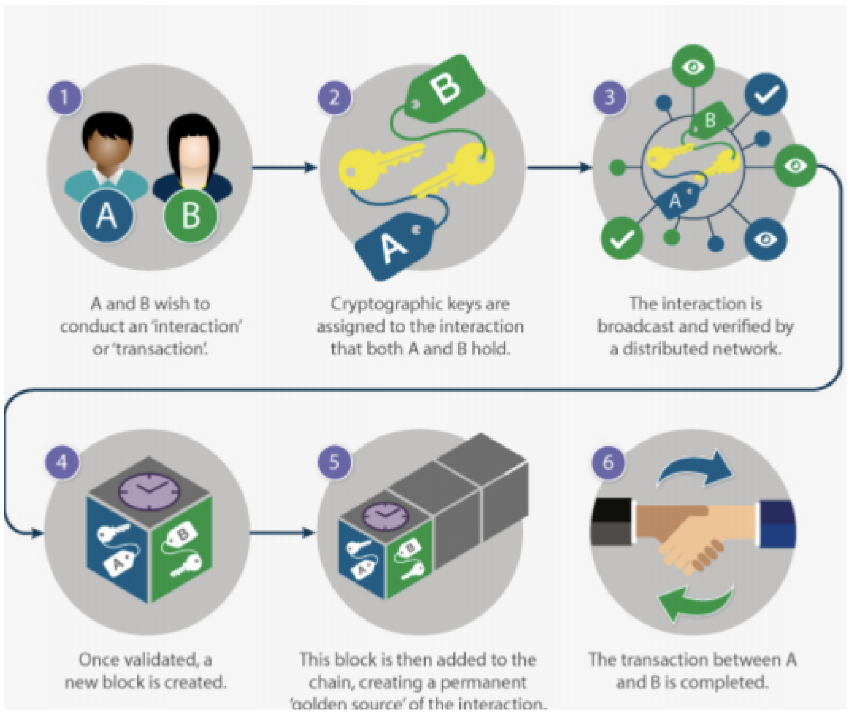
1
A
B
A and B wish to conduct an 'interaction' or 'transaction'.
2
B
A
Cryptographic keys are assigned to the interaction that both A and B hold.
3
B
A
The interaction is broadcast and verified by a distributed network.
4
B
A
Once validated, a new block is created.
5
B
A
This block is then added to the chain, creating a permanent 'golden source' of the interaction.
6
The transaction between A and B is completed.

Figure 26.9: How Bitcoin works.

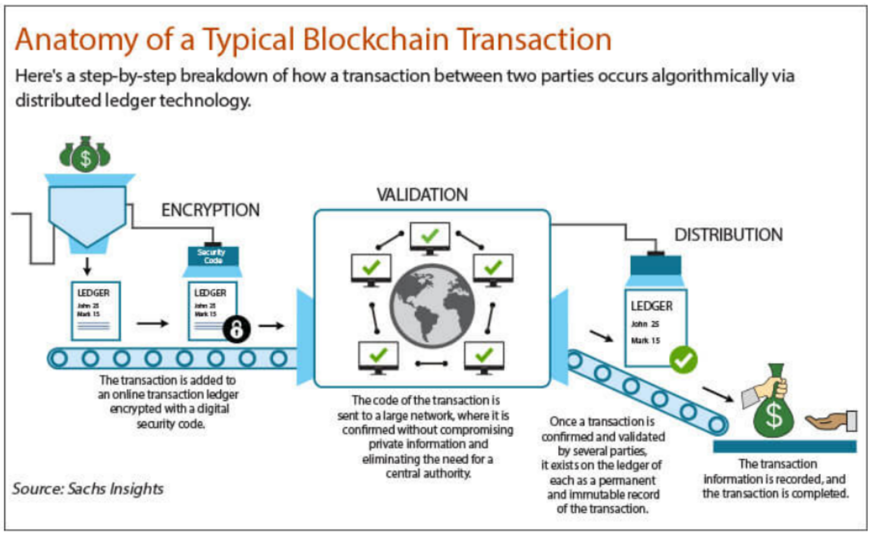
Anatomy of a Typical Blockchain Transaction
Here's a step-by-step breakdown of how a transaction between two parties occurs algorithmically via distributed ledger technology.
ENCRYPTION
Security Code
LEDGER
John 25
Mark 15
The transaction is added to an online transaction ledger encrypted with a digital security code.
VALIDATION
The code of the transaction is sent to a large network, where it is confirmed without compromising private information and eliminating the need for a central authority.
DISTRIBUTION
Once a transaction is confirmed and validated by several parties, it exists on the ledger of each as a permanent and immutable record of the transaction.
The transaction information is recorded, and the transaction is completed.
Source: Sachs Insights

Figure 26.10: How blockchain works.