**1.Hashing algorithm - at what scale do they work?**

A hash function is any function that can be used to map data of arbitrary size to fixed-size values. The values returned by a hash function are called hash values.

In Video Integrity Verification we are using four such algorithms.

1.SHA 512

2.HMAC 256

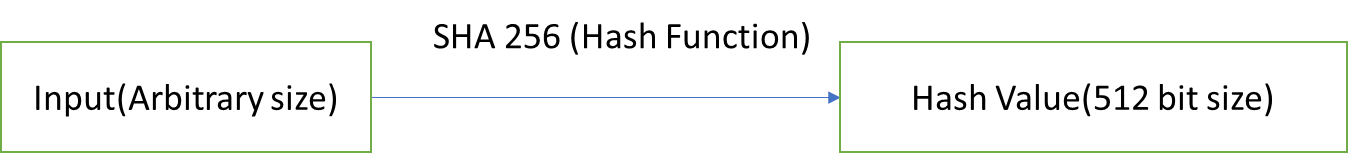
3.HMAC 512

4.EdDSA

**SHA 512**

SHA-512 generates an almost-unique 512-bit (32-byte) signature for a text or a data file.

SHA-512 is a function of cryptographic algorithm SHA-2, which is an evolution of famous SHA-1. SHA-512 is very close to Sha-256 except that it used 1024 bits "blocks", and accept as input a 2^128 bits maximum data length (4.25352959 × 1025 terabytes). SHA-512is also very secure and could be useful on CPU's with 64 bit operations.

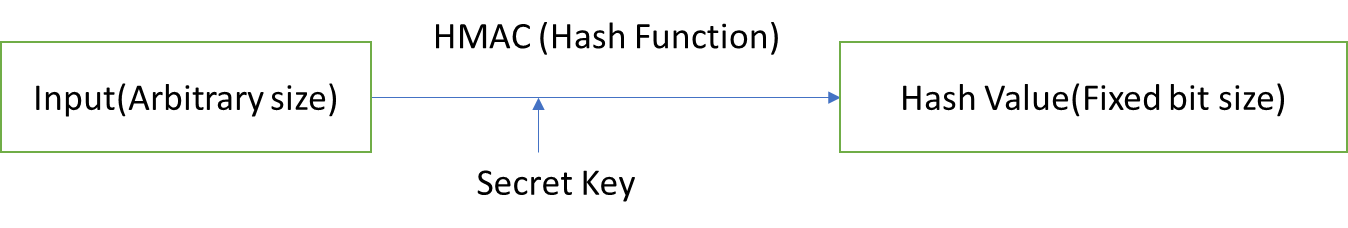


**HMAC 256, HMAC 512**

In cryptography, an HMAC (sometimes expanded as either keyed-hash message authentication code or hash-based message authentication code) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key. As with any MAC, it may be used to simultaneously verify both the data integrity and the authenticity of a message.

HMAC 256 or HMACSHA256 is a type of keyed hash algorithm that is constructed from the SHA-256 hash function and used as a Hash-based Message Authentication Code (HMAC). The HMAC process mixes a secret key with the message data, hashes the result with the hash function, mixes that hash value with the secret key again, and then applies the hash function a second time. The output hash is 256 bits in length. SHA-256 is faster than SHA-512 on 8, 16 and 32 bit machines.

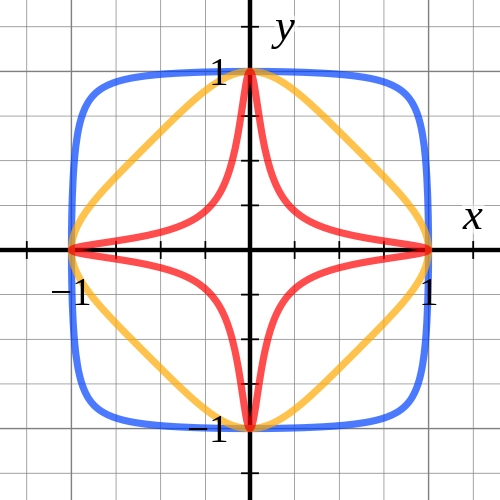
HMACSHA512 HMAC 512 is a type of keyed hash algorithm that is constructed from the SHA-512 hash function and used as a Hash-based Message Authentication Code (HMAC). The HMAC process mixes a secret key with the message data and hashes the result. The output hash is 512 bits in length.SHA-512 is faster than SHA-256 on 64 bit machines (as they use 64 bit arithmetic internally).



4.EdDSA

In public-key cryptography, Edwards-curve Digital Signature Algorithm (EdDSA) is a digital signature scheme using a variant of Schnorr signature based on twisted Edwards curves.

In mathematics, the Edwards curves are a family of elliptic curves studied by Harold Edwards in 2007. The concept of elliptic curves over finite fields is widely used in elliptic curve cryptography. Applications of Edwards curves to cryptography were developed by Daniel J. Bernstein and Tanja Lange: they pointed out several advantages of the Edwards form in comparison to the more well known Weierstrass.

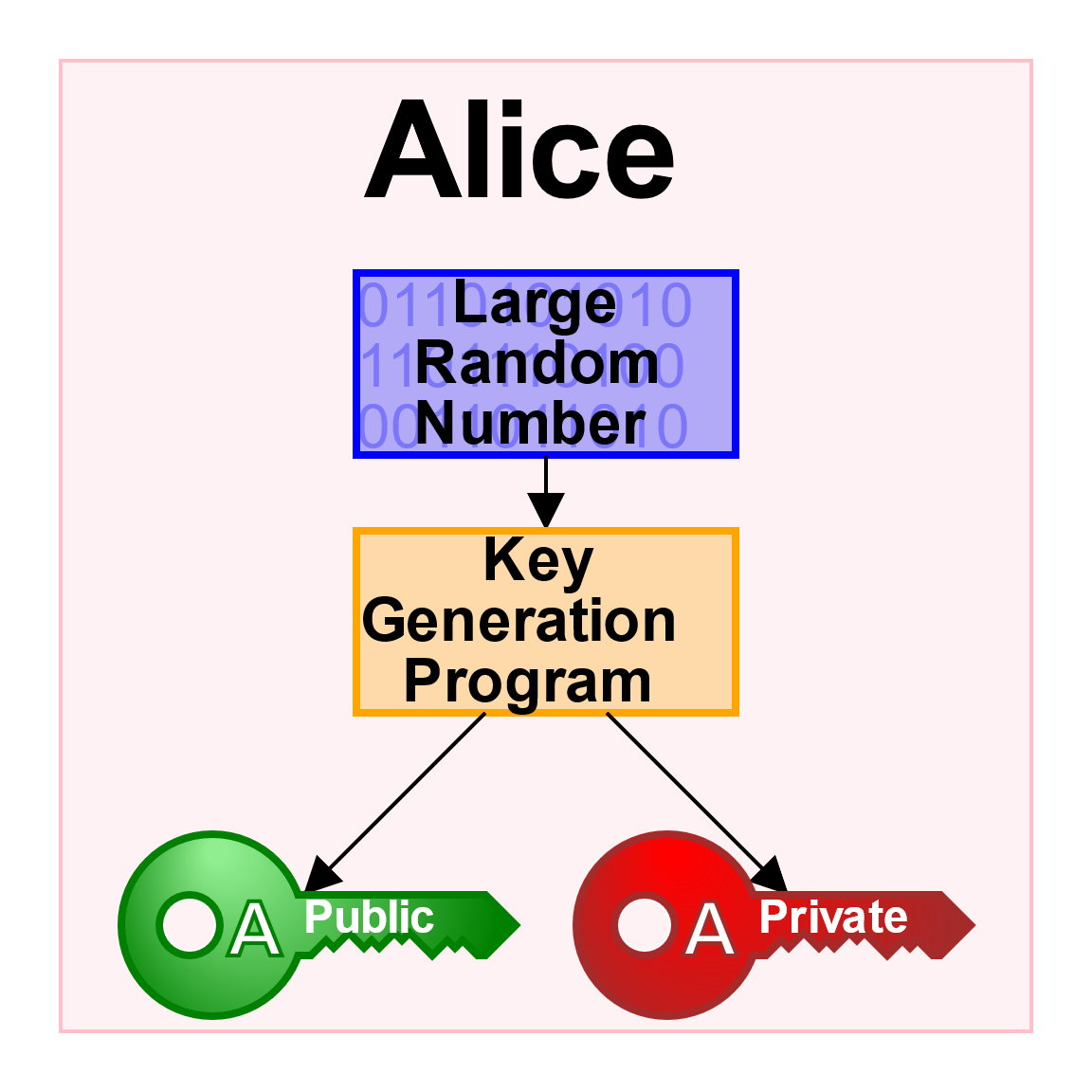


Edwards curves of equation x2 + y2 = (1 − d )\*X\*Y

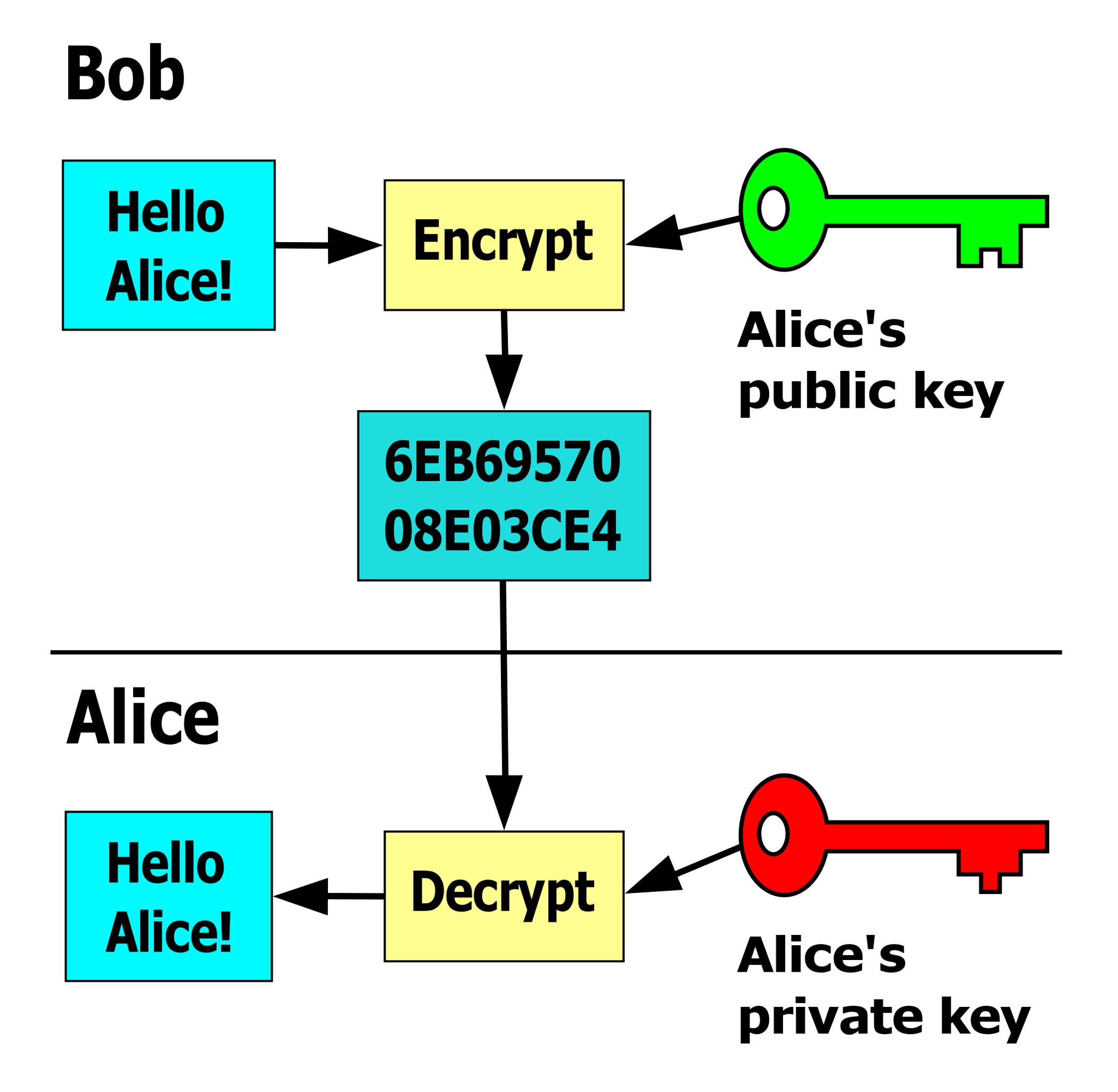
A digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents. A valid digital signature, where the prerequisites are satisfied, gives a recipient very strong reason to believe that the message was created by a known sender (authentication), and that the message was not altered in transit (integrity).

Digital signatures are a standard element of most cryptographic protocol suites, and are commonly used for software distribution, financial transactions, contract management software, and in other cases where it is important to detect forgery or tampering.

Public-key cryptography, or asymmetric cryptography, is a cryptographic system that uses pairs of keys: public keys, which may be disseminated widely, and private keys, which are known only to the owner. The generation of such keys depends on cryptographic algorithms based on mathematical problems to produce one-way functions. Effective security only requires keeping the private key private; the public key can be openly distributed without compromising security.



In such a system, any person can encrypt a message using the receiver's public key, but that encrypted message can only be decrypted with the receiver's private key.



Robust authentication is also possible. A sender can combine a message with a private key to create a short digital signature on the message. Anyone with the sender's corresponding public key can combine the same message and the supposed digital signature associated with it to verify whether the signature was valid, i.e. made by the owner of the corresponding private key.

**2.What are Block chain concepts for VIDEO identifying tampering?**

We are storing a hash value of the video into the blockchain and named it as video integrity code (VIC value) which is unique for each video file. VIC value of a video is stored in a block along with the previous hash (which is the hash value of the previous block data - call it as “block HMAC”) and timestamp. So that it generates an unbreakable chain of video file hashes in chronological order. First block stored in blockchain is called as Genesis block. Genesis block does not contain a previous hash.

If we tamper/edit a video file, then it will generate a different VIC value. So that the Tampered value does not match with the original Video’s VIC value. In that case, when we try to performing a video hash checking, we can clearly say that video is tampered. But If, the hashes are easily accessible to an attacker, one could tamper with the video and replace its original hash value without leaving any clue of modification. In this case by using concept of blockchain, if each block contains hash of the previous block data, so that we can perform a video integrity check by verifying previous hash stored in current block. Previous hash data in the current block should be verified with the entire data stored in the previous block. For this, we can take the entire previous block of data together (as data for hashing) and current block data (as a key for hashing). For an “Original Video” the resultant value must be equal to block HMAC value stored in the current block. Otherwise the video is identified as tampered.

**3.Background research – literature**

A. Blockchain

A blockchain is known as a distributed open ledger containing a block of transactions executed in a network, and is maintained by a node itself. A block is added to the chain, and the hash of the block is included in the next block. Thus, chronological chain of data guaranteed by the sequential nested blocks states that the data could not be changed without changing its block and the following blocks. The individual block is recognized by the cryptographic hash on the header of the block, which is a hash of its parent block. Every block containing the hash of its parent links the sequence of blocks to create a chain. The body part of the block contains batches of valid transactions that are hashed and encoded into a Merkle tree. In addition, a timestamp and a nonce value are added to the block, where the nonce is a random integer number that is repeatedly discovered until the hash of the block will contain a run of leading zeros that makes the block qualified to be added to the blockchain. As this iterative process requires time and resources, the calculation of correct nonce constitute proof of work in blockchain. Hence, the integrity of blockchain is based on chained cryptography that makes it quite difficult to break. The blockchain technology, which was originally designed for a financial ledger in a decentralization concept, can be extended to other frameworks and in other contexts, such as medical data access and permission management. However, a full-principle implementation of blockchain requires the system to be decentralized, and a proof of work algorithm to be implemented. In some cases, such as where there is lack of infrastructure, highly sensitive data, or low power devices, integrating a full-principle implementation can be an overwhelming problem.

B.EdDSA

EdDSA (Edwards-curve Digital Signature Algorithm) is a modern and secure digital signature algorithm based on performance-optimized elliptic curves, such as the 255-bit curve Curve25519 and the 448-bit curve Curve448-Goldilocks. The EdDSA signatures use the Edwards form of the elliptic curves (for performance reasons), respectively edwards25519 and edwards448. The EdDSA algorithm is based on the Schnorr signature algorithm and relies on the difficulty of the ECDLP problem.

The EdDSA signature algorithm and its variants Ed25519 and Ed448 are technically described in the RFC 8032.

EdDSA Key Generation

**Ed25519** and **Ed448** use small **private keys** (32 or 57 bytes respectively), small **public keys** (32 or 57 bytes) and **small signatures** (64 or 114 bytes) with **high security level** at the same time (128-bit or 224-bit respectively).

Assume the elliptic curve for the EdDSA algorithm comes with a generator point **G** and a subgroup order ***q*** for the EC points, generated from **G**.

The **EdDSA key-pair** consists of:

* **private key** (integer): ***privKey***
* **public key** (EC point): ***pubKey*** = ***privKey*** \* **G**

The **private key** is generated from a **random integer**, known as ***seed*** (which should have similar bit length, like the curve order). The ***seed*** is first hashed, then the last few bits, corresponding to the curve **cofactor** (8 for Ed25519 and 4 for X448) are cleared, then the highest bit is cleared and the second highest bit is set. These transformations guarantee that the private key will always belong to the same subgroup of EC points on the curve and that the private keys will always have similar bit length (to protect from timing-based side-channel attacks). For **Ed25519** the private key is 32 bytes. For **Ed448** the private key is 57 bytes.

The public key ***pubKey*** is a point on the elliptic curve, calculated by the EC point multiplication: ***pubKey*** = ***privKey*** \* **G** (the private key, multiplied by the generator point **G** for the curve). The public key is encoded as **compressed** EC point: the **y**-coordinate, combined with the lowest bit (the parity) of the **x**-coordinate. For **Ed25519** the public key is 32 bytes. For **Ed448** the public key is 57 bytes.

EdDSA Sign

The **EdDSA signing** algorithm ([RFC 8032](https://tools.ietf.org/html/rfc8032#page-13)) takes as input a text message **msg** + the signer's EdDSA **private key** **privKey** and produces as output a pair of integers {**R**, **s**}. EdDSA signing works as follows (with minor simplifications):

EdDSA\_sign(msg, privKey) --> { R, s }

1. Calculate **pubKey** = **privKey** \* **G**
2. Deterministically generate a secret integer **r** = hash(hash(**privKey**) + **msg**) mod **q** (this is a bit simplified)
3. Calculate the public key point behind **r** by multiplying it by the curve generator: **R** = **r** \* **G**
4. Calculate **h** = hash(**R** + **pubKey** + **msg**) mod **q**
5. Calculate **s** = (**r** + **h** \* **privKey**) mod **q**
6. Return the **signature** { **R**, **s** }

The produced **digital signature** is 64 bytes (32 + 32 bytes) for **Ed25519** and 114 bytes (57 + 57 bytes) for **Ed448**. It holds a compressed point **R** + the integer **s** (confirming that the signer knows the **msg** and the **privKey**).

The **EdDSA signature verification** algorithm ([RFC 8032](https://tools.ietf.org/html/rfc8032#page-13)) takes as input a text message **msg** + the signer's EdDSA **public key** **pubKey** + the EdDSA signature {**R**, **s**} and produces as output a boolean value (valid or invalid signature). EdDSA verification works as follows (with minor simplifications):

EdDSA\_signature\_verify(msg, pubKey, signature { R, s } ) --> valid / invalid

1. Calculate **h** = hash(**R** + **pubKey** + **msg**) mod **q**
2. Calculate **P1** = **s** \* **G**
3. Calculate **P2** = **R** + **h** \* **pubKey**
4. Return **P1** == **P2**

How Does it Work?

During the verification the point **P1** is calculated as: **P1** = **s** \* **G**.

During the signing **s** = (**r** + **h** \* **privKey**) mod **q**. Now replace **s** in the above equation:

* **P1** = **s** \* **G =** (**r** + **h** \* **privKey**) mod **q** \* **G** = **r** \* **G** + **h** \* **privKey** \* **G** = **R** + **h** \* **pubKey**

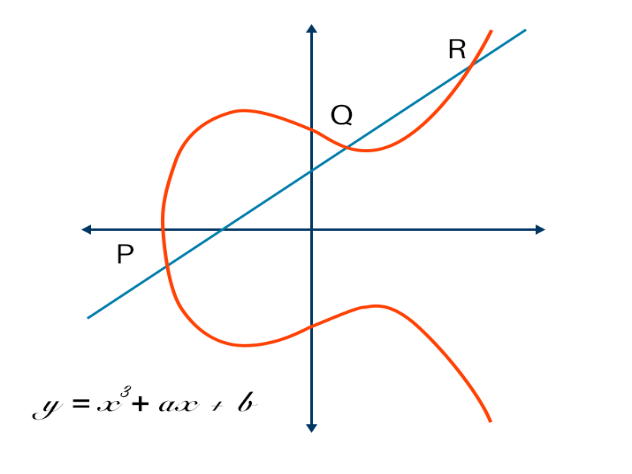
The above is exactly the other point **P2**. If these points **P1** and **P2** are the same EC point, this proves that the point **P1**, calculated by the private key matches the point **P2**, created by its corresponding public key.

**4.How these differ?Elliptical algorithm, Edwards-curve Digital Signature Algorithm**

**(EdDSA)**

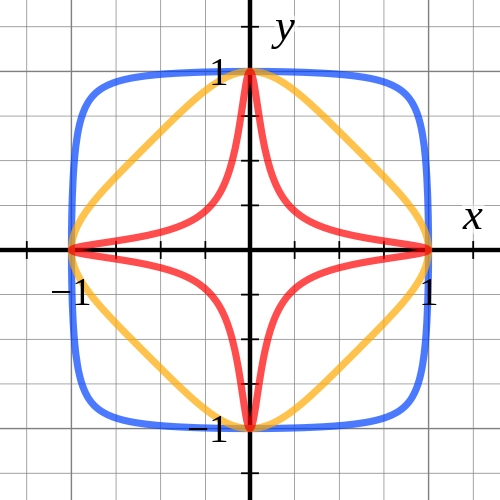
* **ECDSA and EdDSA uses different elliptic-curves.**

The ECDSA is based on the elliptic-curve cryptography (ECC). ECDSA relies on the math of the cyclic groups of elliptic curves over finite fields and on the difficulty of the ECDLP problem (elliptic-curve discrete logarithm problem).



Elliptic curve of equation y=x3+ax+b

EdDSA (Edwards-curve Digital Signature Algorithm) algorithm is based on performance-optimized elliptic curves, such as the 255-bit curve Curve25519 and the 448-bit curve Curve448-Goldilocks. The EdDSA signatures use the Edwards form of the elliptic curves (for performance reasons), respectively edwards25519 and edwards448. The EdDSA algorithm is based on the Schnorr signature algorithm and relies on the difficulty of the ECDLP problem.



Edwards curves of equation x2 + y2 = (1 − d )\*X\*Y

* **ECDSA and EdDSA uses different method for private key generation. We can say that EdDSA uses an improved private key generation process from ECDSA**

The private key of ECDSA is generated as a random integer in the range [0...n-1].

The private key of EdDSA is generated from a random integer, known as seed (which should have similar bit length, like the curve order). The seed is first hashed, then the last few bits, corresponding to the curve cofactor (8 for Ed25519 and 4 for X448) are cleared, then the highest bit is cleared and the second highest bit is set. These transformations guarantee that the private key will always belong to the same subgroup of EC points on the curve and that the private keys will always have similar bit length (to protect from timing-based side-channel attacks). For Ed25519 the private key is 32 bytes. For Ed448 the private key is 57 bytes.

* **ECDSA and EdDSA uses different method for public** **key encoding**

The ECDSA public key pubKey is a point on the elliptic curve, calculated by the EC point multiplication: pubKey = privKey \* G (the private key, multiplied by the generator point G).

The public key EC point {x, y} can be compressed to just one of the coordinates + 1 bit (parity). For the secp256k1 curve, the private key is 256-bit integer (32 bytes) and the compressed public key is 257-bit integer (~ 33 bytes).

The EdDSA public key pubKey is a point on the elliptic curve, calculated by the EC point multiplication: pubKey = privKey \* G (the private key, multiplied by the generator point G for the curve). The public key is encoded as compressed EC point: the y-coordinate, combined with the lowest bit (the parity) of the x-coordinate. For Ed25519 the public key is 32 bytes. For Ed448 the public key is 57 bytes.

* **ECDSA and EdDSA uses different algorithm for signing and signature verification**

The EdDSA signing algorithm (RFC 8032) is different from The ECDSA signing algorithm (RFC 6979). But both takes as input a text message msg + the signer's private key (privKey) and produces as output a pair of integers {R, s}.

The EdDSA signature verification algorithm (RFC 8032) is different from The ECDSA signature verification algorithm (RFC 6979).

But both takes as input the signed message msg + the signature {r, s} produced from the signing algorithm + the public key (pubKey), corresponding to the signer's private key. The output is boolean value: valid or invalid signature.

5.Draw two diagrams(plan for doing video integrity ) & Block chain diagram - How you would use this approach.

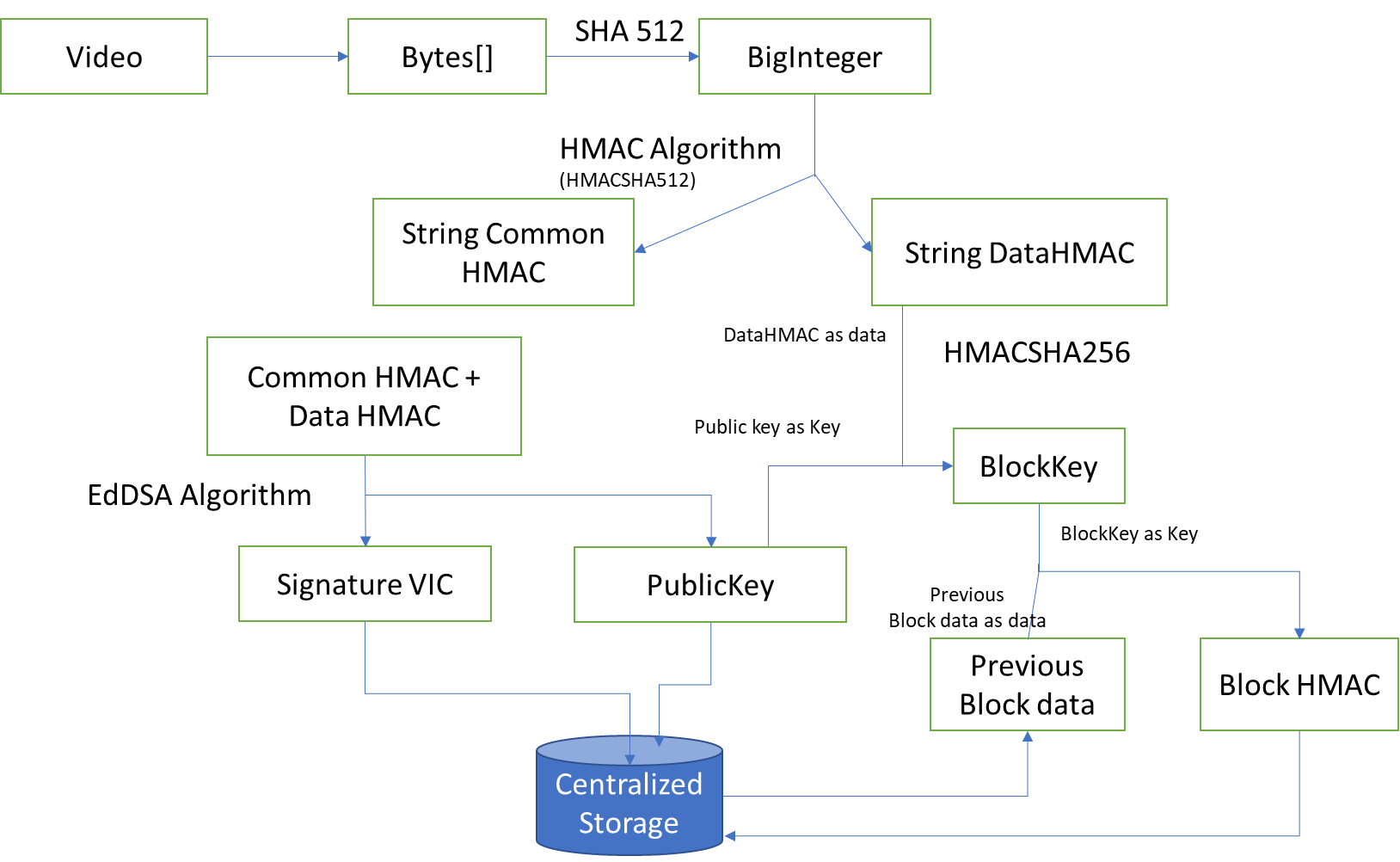


Fig 1. plan for doing video integrity

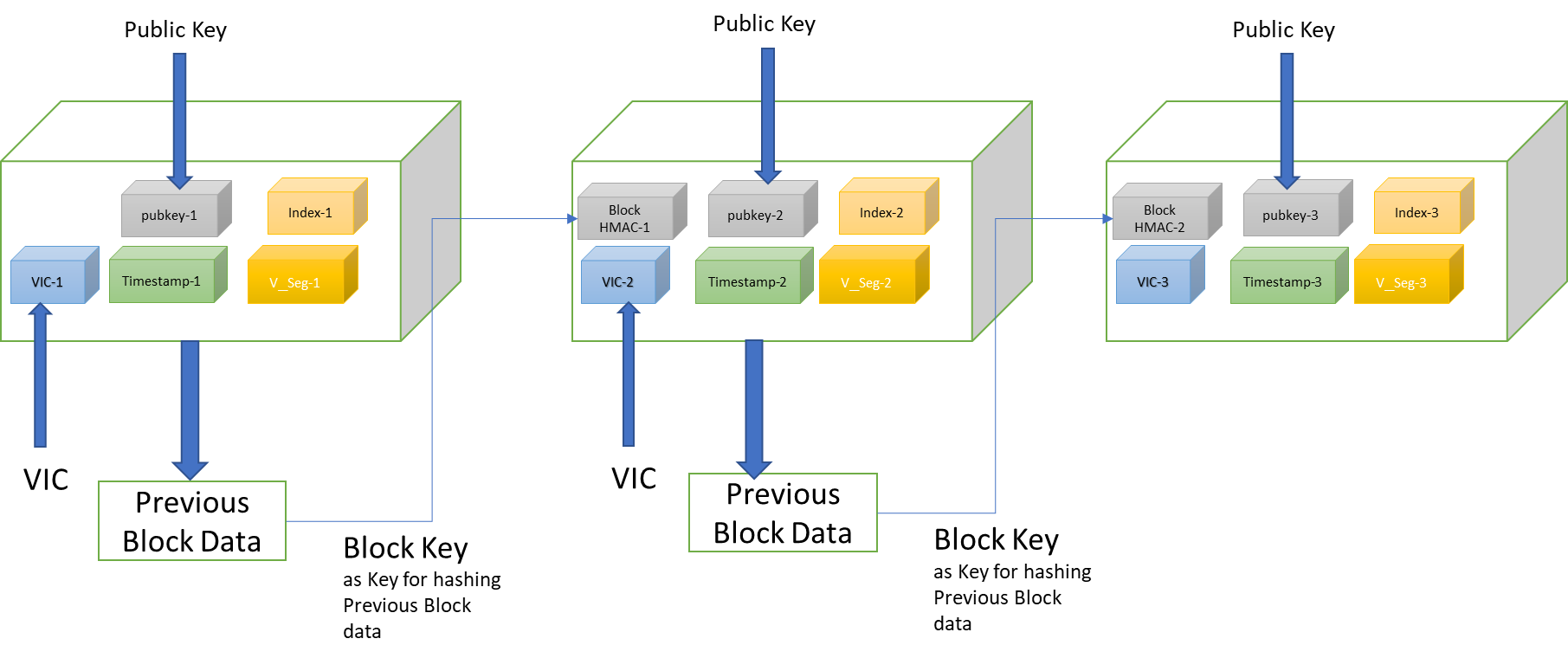


Fig 2. Block chain diagram