A **cryptographic hash function** is a special class of hash function that has certain properties which make it suitable for use in cryptography. It is a mathematical algorithm that maps data of arbitrary finite size to a bit string of a fixed size (a hash function) which is designed to also be a one-way function, that is, a function which is infeasible to invert.
The only way to recreate the input data from an ideal cryptographic hash function's output is to attempt a brute-force search of possible inputs to see if they produce a match.
The input data is often called the *message*, and the output (the *hash value* or *hash*) is often called the *message digest* or simply the *digest*.
From <https://en.wikipedia.org/wiki/Cryptographic_hash_function>

$$M - \text{message} \quad \text{finite length}$$
$$h = H(M)$$

$$|h| = 256 \text{ bits}$$
$$\lfloor h \rfloor = 28 \text{ bits}$$
$$= 7 \text{ hex numb.}$$

$$0000_b = 0_h \equiv 0_d$$
$$0001_b = 1_h \equiv 1_d$$
$$0010_b = 2_h \equiv 2_d$$
$$1001_b = 9_h \equiv 9_{10}$$
$$1010_b = A_h \equiv 10_{10}$$
$$1110_b = E_h \equiv 14_{10}$$
$$1111_b = F_h = 15_{10}$$

Cryptographic hash functions have many information-security applications, notably in digital signatures, message authentication codes (HMACs), and other forms of authentication. They can also be used as ordinary hash functions, to index data in hash tables, for fingerprinting, to detect duplicate data or uniquely identify files, and as checksums to detect accidental data corruption. Indeed, in information-security contexts, cryptographic hash values are sometimes called (*digital*) *fingerprints*, *message digest* or just *hash values*, even though all these terms stand for more general functions with rather different properties and purposes.
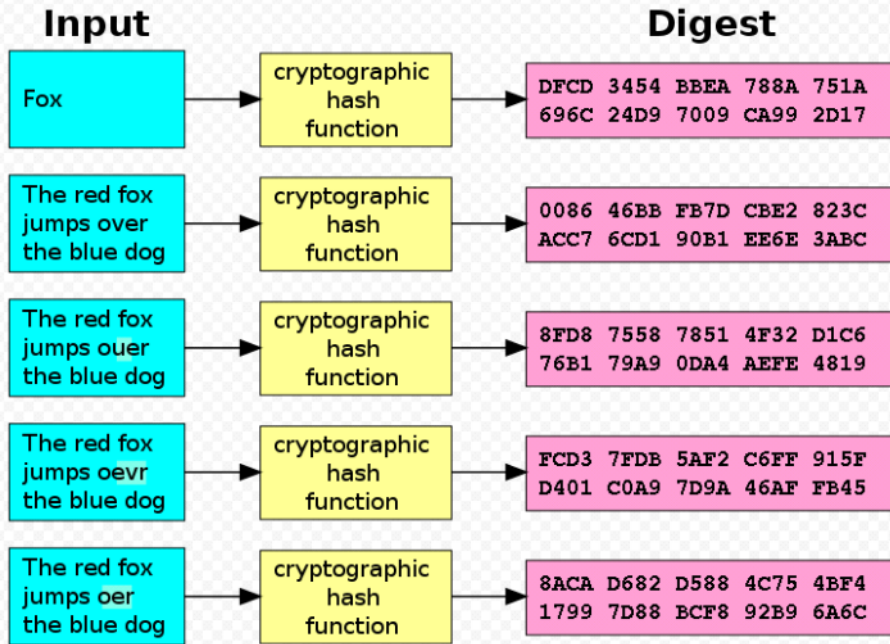
$$M - \text{message} ; \quad H(M) = h$$
$$M \in \{0,1\}^* ; \quad h \in \{0,1\}^{256} ; \quad H : \{0,1\}^* \to \{0,1\}^{256} \; // \text{SHA256}$$

Preimage   Image

M_2 M_1 M_3 M h_1 h
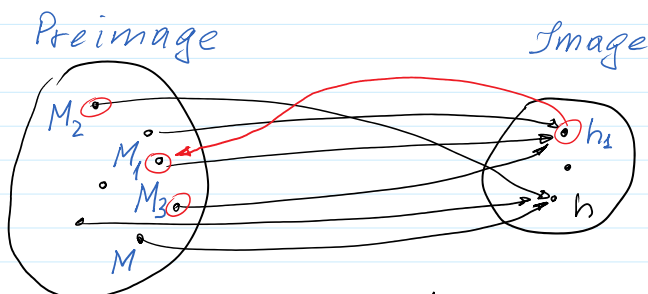
$$1 GB \longrightarrow 256 \text{ bits}$$
many-to-one
$$H(M_1) = H(M_2) = H(M_3) = \ldots = h_1$$

For given h_1 it is infeasible to find any M_i satisfying:

$$H(M_i) = h_1$$

40 Hex numbers = 160 bits

SHA-1

SHA-1: $\{0,1\}^* \to \{0,1\}^{160}$

Avelange effect

$2^{160} \xrightarrow[\text{Par.}]{\text{Birtolay}} 2^{80}$

A cryptographic hash function (specifically SHA-1) at work. A small change in the input (in the word "over") drastically changes the output (digest). This is the so-called avalanche effect.

Properties

- It is quick to compute the hash value for any given finite message.
- A small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value.
- Security properties presented below.

Most cryptographic hash functions are designed to take a string of any finite length as input and produce a fixed-length hash value.
A cryptographic hash function must be able to withstand all known types of cryptanalytic attack.
In theoretical cryptography, the security level of a cryptographic hash function has been defined using the following properties:

- ***Pre-image resistance***
  Given a hash value ***h*** it should be difficult to find any message ***M*** such that ***h = H(M)***. This concept is related to that of one-way function. Functions that lack this property are vulnerable to first preimage attacks.

- ***Second pre-image resistance***
  Given an input M_1 it should be difficult to find (different) input M_2 such that $H(M_1) = H(M_2)$. Functions that lack this property are vulnerable to second-preimage attacks.
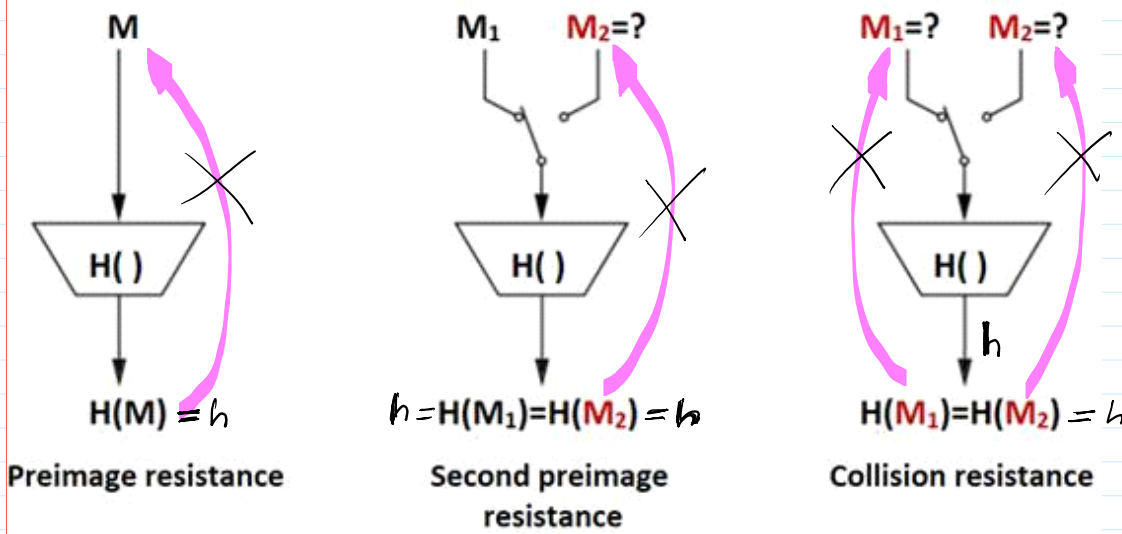
- ***Collision resistance***

It should be difficult to find <mark>any</mark> two different messages M_1 and M_2 such that $H(M_1) = M(M_2)$.
Such a pair is called a cryptographic hash collision. This property is sometimes referred to
as *strong collision resistance.* It requires a hash value at least twice as long as that required for
preimage-resistance; otherwise **collisions** may be found by a birthday attack.[2]
These properties form a hierarchy, in that collision resistance implies second pre-image
resistance, which in turns implies pre-image resistance, while the converse is not true in
general. [3]
The weaker assumption is always preferred in theoretical cryptography, but in practice, a hash-
functions which is only second pre-image resistant is considered insecure and is therefore not
recommended for real applications.
Informally, these properties mean that a malicious adversary cannot replace or modify the input
data without changing its digest.
Thus, if two strings have the same digest, one can be very confident that they are identical.

M M_1 $M_2=?$ $M_1=?$ $M_2=?$

H( )   H( )   H( )

$H(M) = h$ $h = H(M_1) = H(M_2) = h$ $H(M_1) = H(M_2) = h$

Preimage resistance   **Second preimage**   **Collision resistance**
                          resistance

Loan contract

$M_1 = 1000\, €$ loan. M_1 – is a valid loan contract.
$$h = H(M_1);\quad |h| = 256 \text{ bits}$$

$A: PrK_A = x \longrightarrow sign(PrK_A, h) = \sigma_h = (r_A, s_A)$

$\underline{M_1, \sigma_h, PuK_A} \longrightarrow$ I_0 : finds second preimage M_2
                                to create loan contract for $100000\, €$
                                such that $H(M_1) = H(M_2) = h$

Then the valid signature on M_1 is valid also on M_2.

M_2, σ_h claim to Alice to have

$$A: \quad h = H(M_2)$$

$$\text{Verf}(\sigma_h, h, PuK_A) = True \quad \xleftarrow{\quad M_2, \sigma_h \quad} \quad \text{claim to Alice to pay}$$
$$100\,000\ \text{€ (insted of } 1000\text{€)}$$

Illustration

$$nonce = 737327631$$

>> sha256('RootHash PrevHash 737327631')
ans = F4AE534CD226FAF799 8C8424B348E020BA80639A687E93A0B8C5130ED C51E6DE

$$h28('\ldots')$$

C51E6DE

>> sha256('RootHash PrevHash 737327632')
ans = B856211DF2EE15E30AB770C1A43CE014ECFE573182AFD885B28D96854DBC5F21
>> sha256('RootHash PrevHash 737327633')
ans = 9C18C764E347A58E57AC3F7A3C2874D5889A0E802699FEA47EEFF8C03BFEDA69

$$0_h \equiv 0000_2 \quad ; \quad F_h \equiv 1111_2$$

$$h28('\ldots') \twoheadrightarrow 7 \text{ hex numb.}$$
$$hd28('\ldots') \rightarrow \text{decimal num.}$$

Commitment

An illustration of the potential use of a cryptographic hash is as
follows: Alice poses a tough math problem to Bob and claims she has solved it.
Bob would like to try it himself, but would yet like to be sure that Alice is not
bluffing.

$$P = NP$$
$$P \neq NP$$

Therefore, Alice writes down her solution, computes its hash and tells Bob the
hash value (whilst keeping the solution secret).
Then, when Bob comes up with the solution himself a few days later, Alice can
prove that she had the solution earlier by revealing it and having Bob hash it
and check that it matches the hash value given to him before. (This is an
example of a simple commitment scheme; in actual practice, Alice and Bob will
often be computer programs, and the secret would be something less easily
spoofed than a claimed puzzle solution).

Verifying the integrity of files or messages
Main article: File verification
An important application of secure hashes is verification of message integrity.
Determining whether any changes have been made to a message (or a file),
for example, can be accomplished by comparing message digests calculated
before, and after, transmission (or any other event).

$$f \neq f'$$
$$H(f) \neq H(f')$$
$$h \qquad h'$$

For this reason, most digital signature algorithms only confirm the
authenticity of a hashed digest of the message to be "signed". Verifying the
authenticity of a hashed digest of the message is considered proof that the
message itself is authentic.

$$Sign(PrK, h) \neq Sign(PrK, h')$$

[MD5](), [SHA1](), or [SHA2]() hashes are sometimes posted along with files on websites or forums to allow verification of integrity.[6] This practice establishes a [chain of trust]() so long as the hashes are posted on a site authenticated by [HTTPS]().

Password verification[[edit]()]
Main article: [password hashing]()
A related application is [password]() verification (first invented by [Roger Needham]()). Storing all user passwords as [cleartext]() can result in a massive security breach if the password file is compromised. One way to reduce this danger is to only store the hash digest of each password. To authenticate a user, the password presented by the user is hashed and compared with the stored hash. (Note that this approach prevents the original passwords from being retrieved if forgotten or lost, and they have to be replaced with new ones.) **The password is often concatenated with a random, non-secret [salt]() value** before the hash function is applied. The salt is stored with the password hash. Because users have different salts, it is not feasible to store tables of [precomputed]() hash values for common passwords. [Key stretching]() functions, such as [PBKDF2](), [Bcrypt]() or [Scrypt](), typically use repeated invocations of a cryptographic hash to increase the time required to perform [brute force attacks]() on stored password digests.
In 2013 a long-term [Password Hashing Competition]() was announced to choose a new, standard algorithm for password hashing.

Proof-of-work
Main article: [Proof-of-work system]()
A proof-of-work system (or protocol, or function) is an ***economic*** measure to deter [denial of service]() attacks and other service abuses such as spam on a network by requiring some work from the service requester, usually meaning processing time by a computer. A key feature of these schemes is their asymmetry: the work must be moderately hard (but feasible) on the requester side but easy to check for the service provider. One popular system — used in [Bitcoin mining]() and [Hashcash]() — **uses partial hash inversions to prove that work was done,** as a good-will token to send an e-mail. The sender is required to find a message whose hash value begins with a number of zero bits. The average work that sender needs to perform in order to find a valid message is exponential in the number of zero bits required in the hash value, while the recipient can verify the validity of the message by executing a single hash function. For instance, in Hashcash, a sender is asked to generate a header whose 160 bit SHA-1 hash value has the first 20 bits as zeros. The sender will *on average* have to try 2^{19} times to find a valid header.

$$2^{20} = 1M$$

File or data identifier
A message digest can also serve as a means of reliably identifying a file; several [source code management]() systems, including [Git](), [Mercurial]() and [Monotone](), use the [sha1sum]() of various types of content (file content, directory trees, ancestry information, etc.) to uniquely identify them. Hashes are used to identify files

on [peer-to-peer](#) [filesharing](#) networks.

Pseudorandom generation and key derivation
Hash functions can also be used in the generation of [pseudorandom](#) bits, or to [derive new keys or passwords](#) from a single secure key or password.

As of 2009, the two most commonly used cryptographic hash functions were [MD5](#) and [SHA-1](#). However, a successful attack on MD5 broke [Transport Layer Security](#) in 2008.

In February 2005, an attack on SHA-1 was reported that would find collision in about 2^{69} hashing operations, rather than the 2^{80} expected for a 160-bit hash function. In August 2005, another attack on SHA-1 was reported that would find collisions in 2^{63} operations. Though theoretical weaknesses of SHA-1 exist,[14][15] no collision (or near-collision) has yet been found. Nonetheless, it is often suggested that it may be practical to break within years, and that new applications can avoid these problems by using later members of the SHA family, such as [SHA-2](#).

$$1K = 2^{10} = 1024$$
$$1M = 2^{20}$$
$$1G = 2^{30}$$
$$1T = 2^{40}$$
$$SHA1(M) = h$$
$$|h| = 160 \text{ bits}$$

According to birthday paradox it is not required total scan all 2^{160} variants of message M. It is enough to scan $\sqrt{2^{160}} = 2^{80}$ variants.

SHA-2 (Secure Hash Algorithm 2) is a set of [cryptographic hash functions](#) designed by the United States [National Security Agency](#) (NSA).[3]

SHA-2 includes significant changes from its predecessor, [SHA-1](#).
The SHA-2 family consists of six hash functions with [digests](#) (hash values) that are 224, 256, 384 or 512 bits:
SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.

h 28

However, to ensure the long-term robustness of applications that use hash functions, there was a [competition](#) to design a replacement for SHA-2.
On October 2, 2012, Keccak was selected as the winner of the [NIST](#) hash function competition.
A version of this algorithm became a [FIPS](#) standard on August 5, 2015 under the name [SHA-3](#).

HMAC - H Message Authentication Code

Use in building other cryptographic primitives: symmetric e-signature realization
Hash functions can be used to build other cryptographic primitives.
For these other primitives to be cryptographically secure, care must be taken to build them correctly.

Message authentication codes (MACs) (also called keyed hash functions) are often

Information Authentication

For these other primitives to be cryptographically secure, care must be taken to build them correctly.

Message authentication codes (MACs) (also called keyed hash functions) are often built from hash functions. HMAC is such a MAC.

[handwritten: → Authentication / Integrity]

Keyed-hash message authentication code (HMAC) is a specific type of message authentication code (MAC) involving a cryptographic hash function (hence the 'H') in combination with a secret cryptographic key.

As with any MAC, it may be used to *simultaneously* verify both the *data integrity* and the *authentication* of a message.

Any cryptographic hash function, may be used in the calculation of an HMAC.

The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and on the size and quality of the key.

HMAC based e-signature
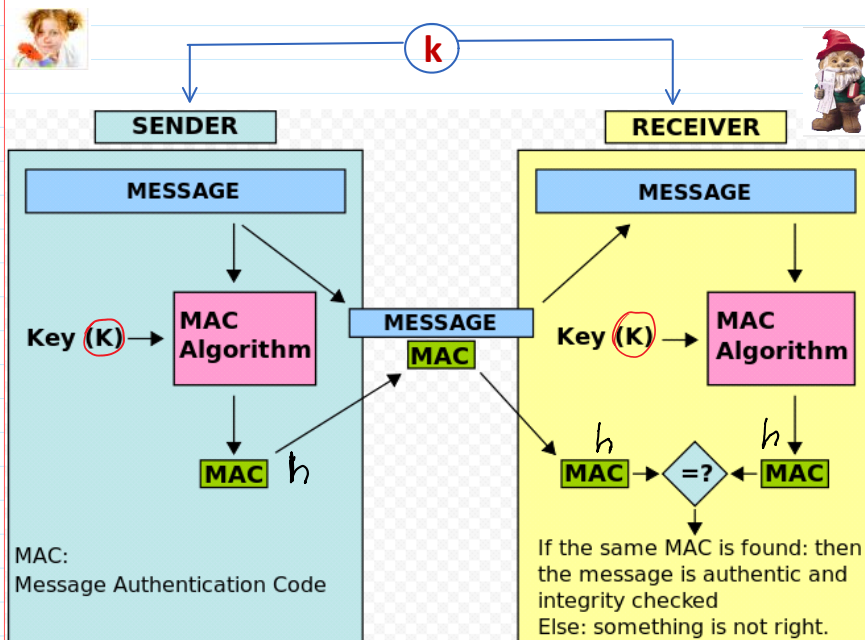
[handwritten: Authentic / Integral]

Till this place

Įdiegti šiuos .m failus į Octave,   išzipuojant failą iš http://crypto.fmf.ktu.lt/xdownload/

h26	2020.03.22 17:08
H26d	2019.10.26 14:30
h28	2020.03.22 17:08
H28d	2019.10.23 22:42
hd26	2020.03.22 17:08
☑ hd28	2020.03.22 17:08

• Octave_Stud 2020.03.22.7z

```
>> sha256('RootHash PrevHash 737327631')
ans = F4AE534CD226FAF7998C8424B348E020BA80639A687E93A0B8C5130EDC51E6DE
>> h28('RootHash PrevHash 737327631')
ans = C51E6DE
>> hd28('RootHash PrevHash 737327631')
ans = 206694110
>> dec2bin(ans)
ans = 1100010100011110011011011110
>> dec2hex(206694110)
ans = C51E6DE
```

Hash functions based on block ciphers

There are several methods to use a block cipher to build a cryptographic hash function, specifically a one-way compression function.
The methods resemble the block cipher modes of operation usually used for encryption.
Many well-known hash functions, including MD4, MD5, SHA-1 and SHA-2 are built from block-cipher-like components

HMAC can be constructed form the block cipher using cipher block chaining (CBC) mode of operation.

CBC-MAC

Cipher block chaining message authentication code (CBC-MAC) is a technique for constructing a message authentication code from a block cipher. The message is encrypted with some block cipher algorithm in CBC mode to create a chain of blocks such that each block depends on the proper encryption of the previous block.
This interdependence ensures that a change to any of the plaintext bits will cause the final encrypted block to change in a way that cannot be predicted or counteracted without knowing the key to the block cipher.
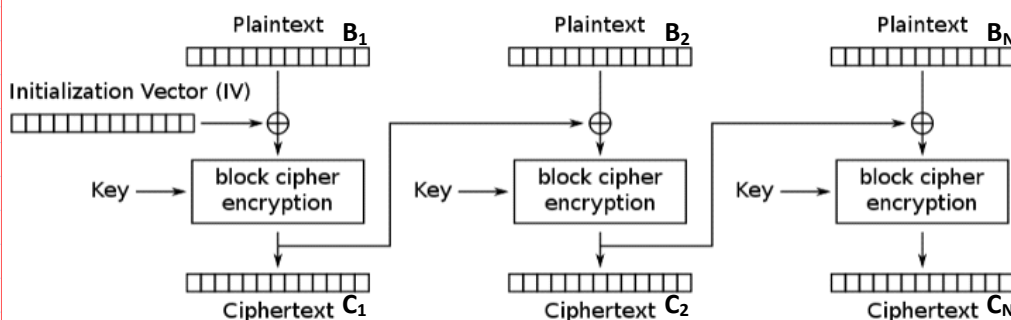From <https://en.wikipedia.org/wiki/CBC-MAC>

$$AES_k - CBC$$

$$M - \text{to be signed.}$$

$$C = AES\_CBC(k, M)$$
$$\downarrow$$
$$256$$

$|k| = 256\,b \Rightarrow B_i = C_i = 256$

M: | B_1 | B_2 | .... | B_N |

$\downarrow AES\_CBC(k,M)$

C: | C_1 | C_2 | ... | C_N |

$hMAC = C_1 \oplus C_2 \oplus ... \oplus C_N = h$
bitwise XORing
$|h| = 256\,b.$

Plaintext B_1 ... Plaintext B_2 ... Plaintext B_N
Initialization Vector (IV)
Key → block cipher encryption
Key → block cipher encryption
Key → block cipher encryption
Ciphertext C_1 ... Ciphertext C_2 ... Ciphertext C_N

Cipher Block Chaining (CBC) mode encryption

Plaintext B_1, Plaintext B_2, Plaintext B_N

Initialization Vector (IV)

Key → block cipher encryption

Ciphertext C_1, Ciphertext C_2, Ciphertext C_N

(handwritten top right) bitwise XORing |h| = 256 b.

****************************** Till this place ****************

Asymmetric Signing - Verification
Public Parameters - PP: >> p = 268 435 019; >> g=2;

$$\mathcal{I}_p^* = \{1, 2, \ldots, p-1\}$$

Signature Creation - Verification
S=Sign(PrK_A, h)= σ =(r, s)
V=Ver(PuK_A, σ, h), V$\in$\{True, False\}$\equiv$\{1, 0\}

Alice
Hello Bob → Sign ← Alice's private key
Hello Bob BE459576 785039E8 $\left.\begin{array}{c}r\\s\end{array}\right\}$ 6

Bob
Hello Bob ← Verify ← Alice's public key

IBM Hyperledger Fabric
IBM Trust Food

Signature creation by sender Alice:
M - any message of finite length to be signed.
1. **M is hashed with h-function H() by computing its h-value   h=H(M)**
2. **Signature is computed on h-value h:**
   σ=Sig(PrK_A, h)=(r,s).

Signature verification:
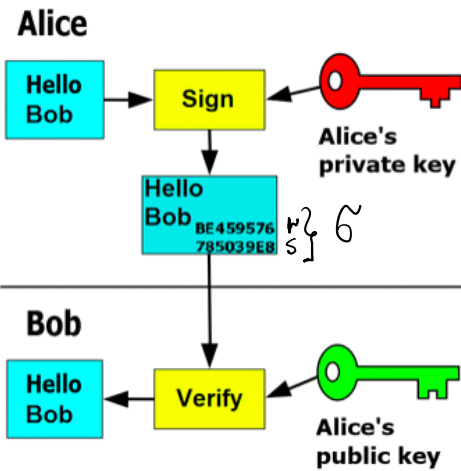1. **Received message M' is hashed by receiver Bob h'=H(M').**
2. **Signatutre is verified by verification function Ver(PuK_A, σ, h').**
   If PrK_A=x, PuK_A=a and $a = g^x \bmod p$
   AND
   If M=M'
   Then signature is valid and V$\in$\{True\}.

Hash and Sign paradigm

******************************** Till this place ****************

M - message to be hashed

k - secret encr. key

$A(k)$ $\qquad$ $B(k)$

A: wish to encr. message m together wit providing its authenticity and integrity.

A $\qquad$ Whatsup $\qquad$ B

K_{A1} $\qquad$ K_{A2}

① K_{B1} $\qquad$ K_{B2}

$$K_{AB} = \underbrace{(K_B)^x \bmod p}_{K_1} = K = \underbrace{(K_A)^y \bmod p}_{K_2} = K_{BA}$$
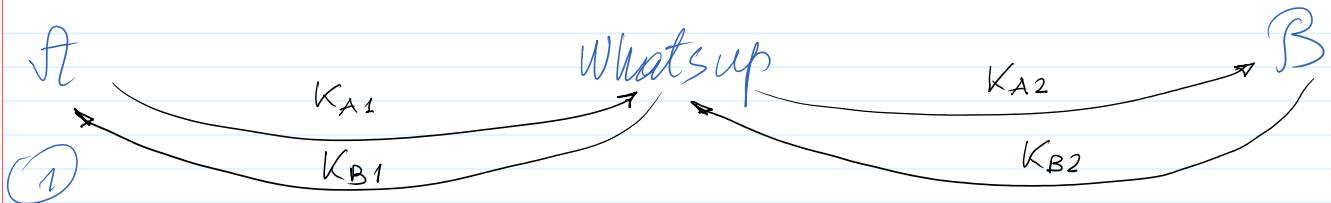
M - message

$Enc(K_1, m) = C_1 \longrightarrow$

$\qquad\qquad Dec(K_1, C_1) = m$

$\qquad\qquad\qquad\qquad$ Data center

$\qquad\qquad\qquad Enc(K_2, m) = C_2 \longrightarrow$

$\qquad\qquad\qquad\qquad Dec(K_2, C_2) = m$

② Whatsup software and its updating

Backdoors $\longrightarrow$

Chosen Plaintext Attack

A:

$E_{CBC}(\boldsymbol{k}, m) = c$ ⎫ encrypt
$H_{CBC}(\boldsymbol{k}, c) = h$ ⎭ & hash $\xrightarrow{\quad c, h \quad}$

B: 1) $H_{CBC}(\boldsymbol{k}, c) = h'$

? $h \overset{?}{=} h'$ if ok

2) $D(\boldsymbol{k}, c) = m$