L2:
Hashing, Cryptographic Hashes, and Password Hashing

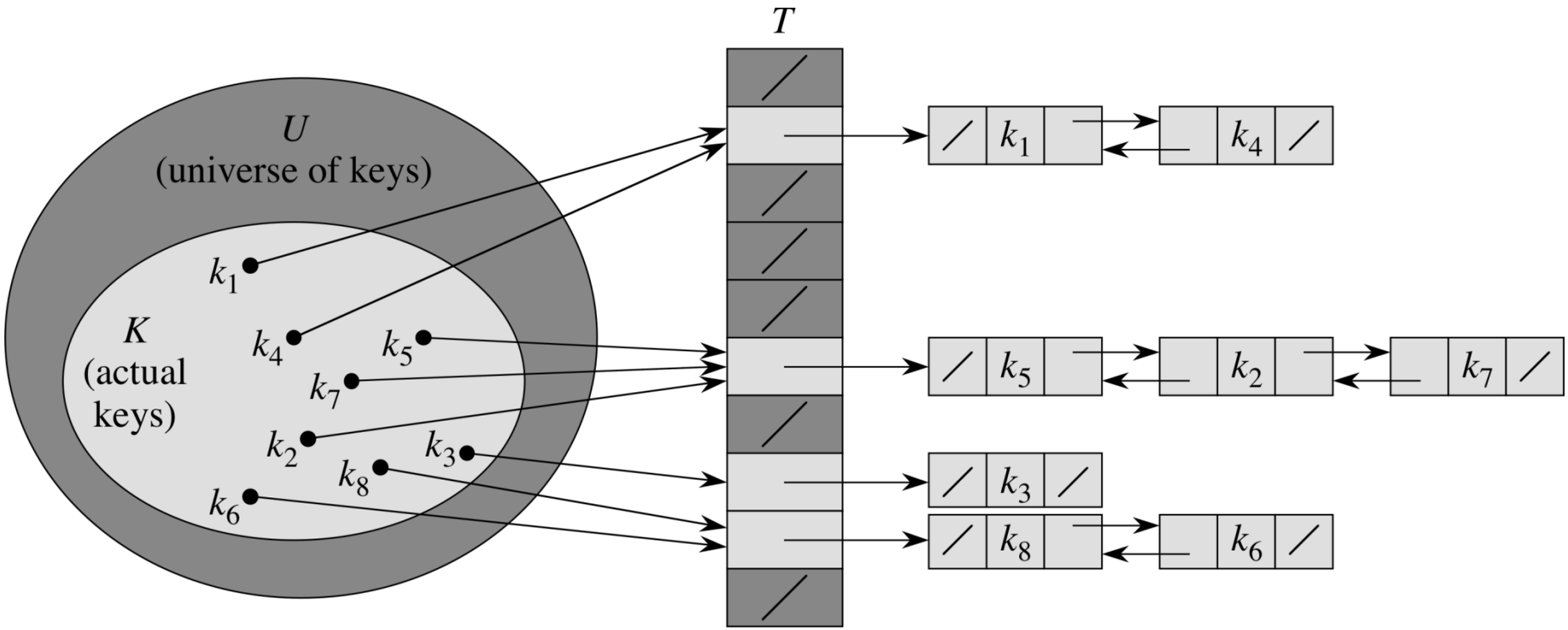
Sudhir Aggarwal and Shiva Houshmand
Florida State University
Department of Computer Science
E-Crime Investigative Technologies Lab
Tallahassee, Florida 32306
August 5-7, 2015

Password Cracking
University of Jyväskylä
Summer School August 2015

Common Data Structures: Hash Tables

- Dynamic set of elements. Want to support insert, search (find), delete.
- Sometimes called a dictionary
- A hash table is useful for this
  - hash tables often use chaining
- Analysis
  - **search**: expected time (under reasonable assumptions) is O(1). Worst case is Θ(n)
  - **insert**: O(1)
  - **delete**: similar to search or O(1)

Hashing with chaining (using linked lists)



Using a hash function

- Universe of possible keys U is large
- set of actual keys stored K is small compared to this (storage is $\Theta(K)$)
- Use a hash function:

$$h: U \rightarrow \{0, 1, \ldots, m-1\}$$

  - k hashes to slot h(k)
  - two or more keys could hash to same slot called a collision
  - collision resolution by chaining. (One could also do open addressing for collision resolution)

Analysis of *Search* operation using hash table with chaining

- n = # of elements in the table
- m = # of slots in the table
- load factor $\alpha = n/m$
- Worst case: all n keys hash to same slot and is thus $\Theta(n)$
- Average case depends on how the hash function distributes the keys

Assumption: simple uniform hashing

- That is, any element is equally likely to hash into any of the m slots.
- For j = 0, 1, …, m -1, let the length of the list T[j] be n_j. Note that n_j is a *random variable*.
- The expected value of this r.v. is $E[n_j] = \alpha = n/m$
- So the expected time of an *unsuccessful* search is simply $\Theta(1 + \alpha)$
- Successful search requires a more complex analysis since we must figure out how many elements were inserted into the list after the searched element. But it turns out to still be $\Theta(1 + \alpha)$
- If n = O(m), then we get that search takes O(1)
- Note that insert take O(1) since we insert at head of queue, and if we have a pointer to the element, then delete takes O(1) also.

Typical classes of hash functions (assuming keys are natural numbers)

- Division or "mod" method

  $h(k) = k \bmod m$

- Multiplication method

  let A be a constant $0 < A < 1$

  $h(k) = \lfloor m(kA \bmod 1) \rfloor$

  note that here mod 1 means get the fractional part of kA.

Example hash function

- Let U be the set of all strings.
- We map a string into an 8 bit ASCII value as follows:

  s → sum of ASCII character values Mod 256

- alpha → (97 + 108 + 112 + 104 + 97) mod 256

  = 518 mod 256 = 6

- Note: h: U → {0, 1, …, 255}

Cryptographic hash functions

- A hash function is a mathematical, efficiently computable function that has fixed size output:
  - $F : \{0, 1\}^N \rightarrow \{0,1\}^n$, where $N > n$
  - $F: \{0, 1\}^* \rightarrow \{0,1\}^n$
- In cryptography, the first type of hash function is often called a compression function, with the name hash function reserved for the unbounded domain type.
- Note: a hash function does not have a key and anyone can compute the same hash from the same message. However *keyed hashes* do use a key

Checksums and CRCs

- Used to provide integrity checks against ***random faults***.
- **Not** sufficient for **protection against *malicious* or *intentional* modification**.
  - Easy to make changes and re-compute the CRC to match.
- In the past, it was believed that the use of CRCs within encryption was sufficient to provide integrity. However, that is no longer considered adequate:
  - Example: The use of CRCs in the WEP protocol resulted in a serious vulnerability, allowing for powerful active attacks.

Cryptographic hash functions (also called message digests)

- The security of hash functions is defined empirically, if the following problems are found to be computationally infeasible:
  - One way:
    - Given y, find x such that h(x) = y
  - Second pre-image resistant:
    - Given x, find y≠ x such that h(x) = h(y)
  - Collision-resistant:
    - Find y, x, with y≠ x such that h(x) = h(y)

  - *Can you prove that collision resistant (also called strong collision resistant implies second pre-image resistant (also called weak collision resistant)?*

One way function: what is computationally infeasible?

- Given y, find x such that h(x) = y
  - An inverse problem
  - If it is a cryptographic hash function with 128 bit digest, need to try about half of the messages to find a message that maps to y.
  - Need to try about 2^{127} messages
  - This should be computationally infeasible.

Second pre-image resistant - computational cost

- Given x, find y≠ x such that h(x) = h(y)
  - Try messages to find some y that maps to the same value as x.
  - For a 128 bit digest, the expected number of messages to try is again about 2^{127} messages for a 0.5 probability of success

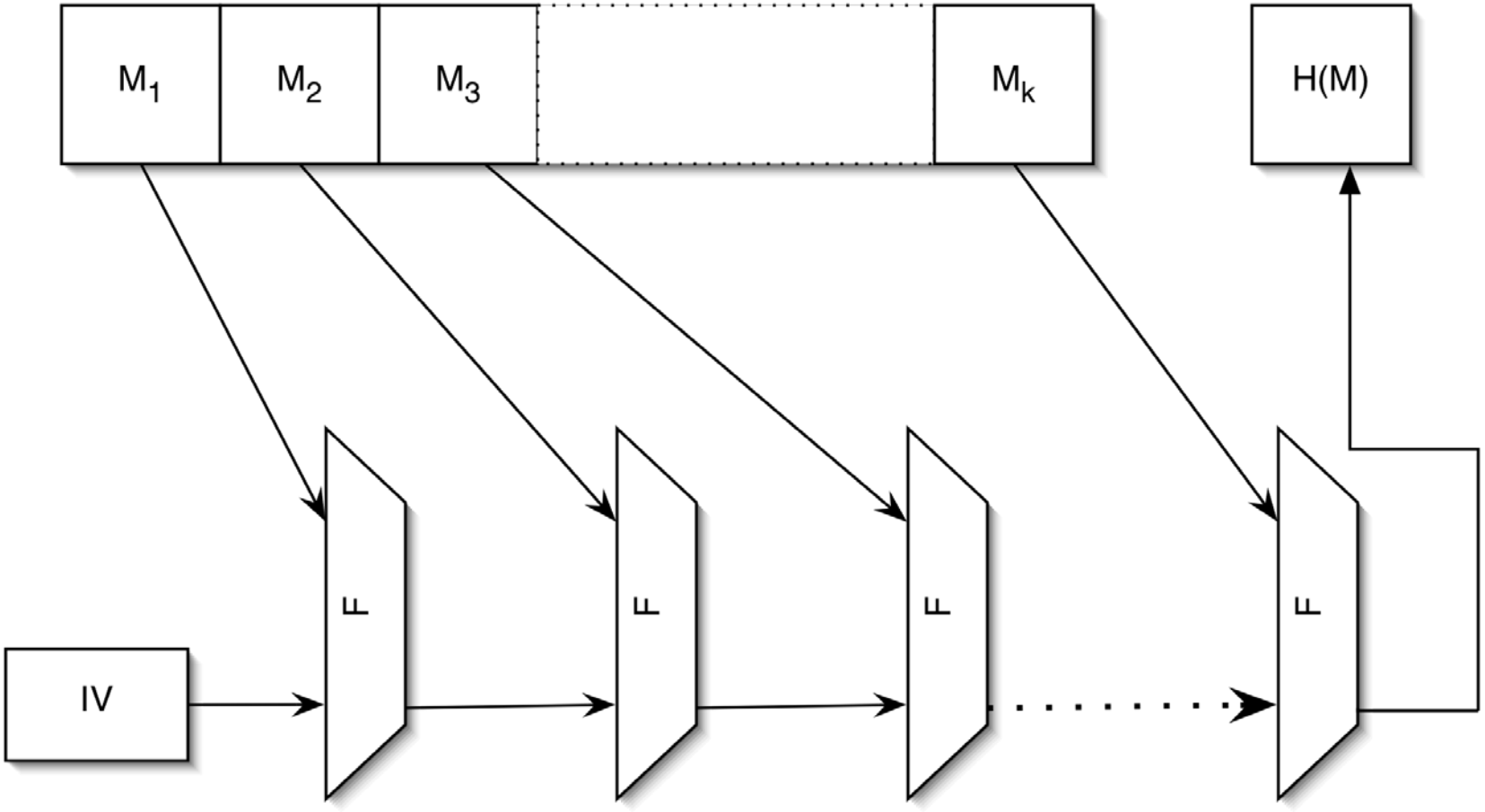
Collision resistant - computational cost

- Find y, x, with y≠ x such that h(x) = h(y)
  - We can show that for a cryptographic hash function, this requires solving the birthday problem, which is about 2^{64} messages for a 128 bit message digest
  - Note that collision resistant
    $\Rightarrow$ second pre-image resistant
  - Use (A$\Rightarrow$B is the same as $\neg$B $\Rightarrow$ $\neg$ A)

Constructing hash functions

- Since constructing secure hash functions is a difficult problem, the following approach has been taken in practice:
  - Construct a good compression function. Since the domain of compression functions are "small" they are easier to test for the desired properties.
- Use the MD construction (next) to turn a one-way, collision-resistant compression function into a hash function with similar properties.

Merkle-Damgard (MD)



Here F() is a compression function, and the MD construction transforms it into a hash function on larger blocks:
H(M_1) = F(IV, M_1), H(M_1 || M_2) = F(H(M_1),M_2) = F(F(IV, M_1), M_2), and so forth

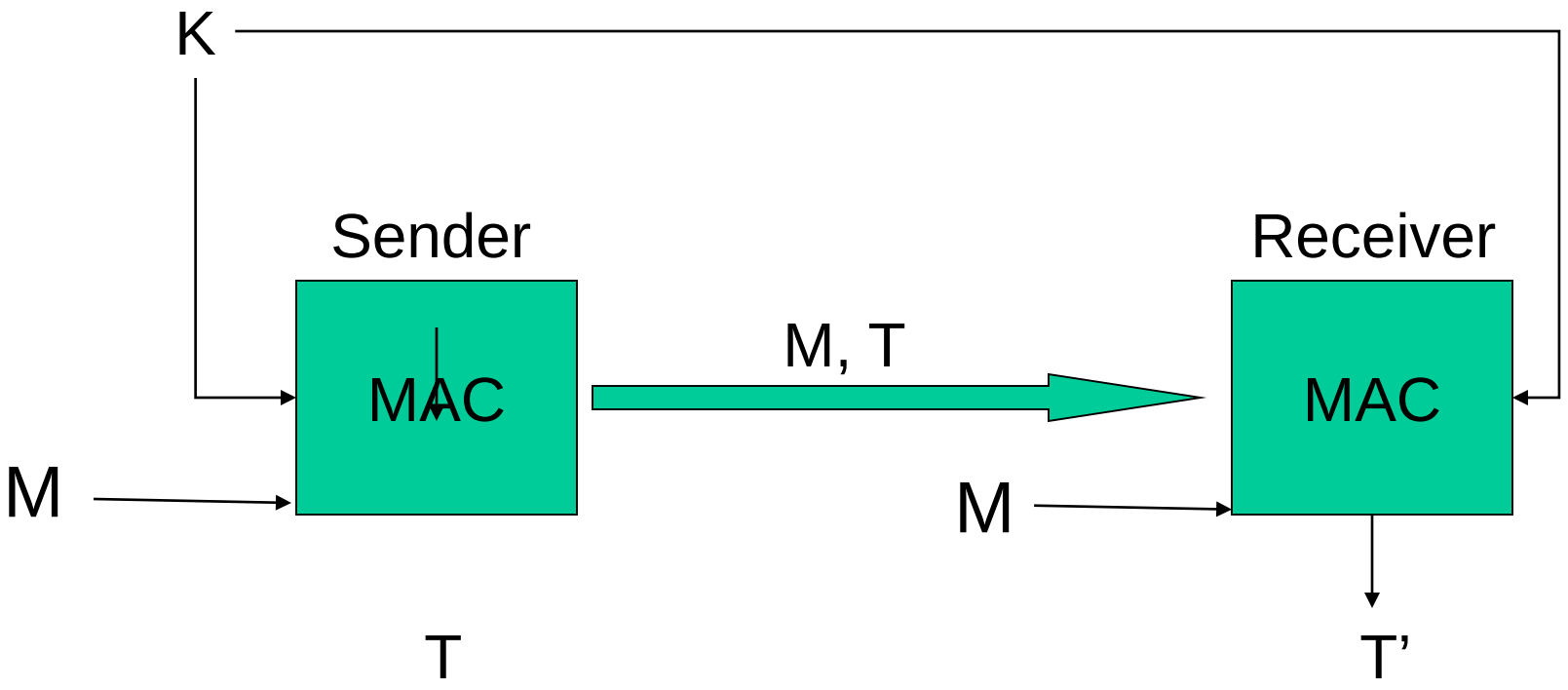
Applications of Hash Functions

- System integrity protection:
- For password verification, eliminating the need to keep passwords
- As building blocks for **message authentication codes** (MACs) and **digital signature** algorithms.

Integrity Protection without MAC (can still do so if you are careful)

- h(K || m) for integrity (know m and h())
  - Concatenate a secret key K with a message m and then use a cryptographic hash function
- h(m || K) (know m and h())
  - Concatenate the secret key K at the end
- Which is better? *The first one is subject to the message extension attack, so the second strategy turns out to be better*

Using MACs



The verification succeeds if the re-computed tag T' equals the original tag T.

Password authentication

- User authenticates by entering a password -- this is checked against the server's database
  - Pros:
    - Supported by almost every system
    - Users familiar with the process
  - Cons:
    - Good password management is crucial
    - Storing passwords securely can be a problem

Printing bytes & ASCII characters

- ASCII is a 7 bit character set (0-127)
  - fits into a byte, remaining bit could be set to 0 or used as a parity bit
  - 95 printable characters, 33 non-printing
- Some printable encodings
  - quoted-printable encoding (QP encoding)
    - printable characters as is
    - non printable characters represented as =$hex_1 hex_2$
  - Base64 encoding (used in Privacy Enhanced Mail)
    - use character set of $2^6 = 64$ (A-Z, a-z, 0-9, +, /)
    - = is used as a special suffice for termination conditions
    - 3 bytes are converted into 4 sets of 6 bit characters
    - the 6 bit value of each set indexes into the character set
    - note that 3 bytes encoded as 4 bytes
    - result (due to termination rules) is output that is a multiple of 4 bytes
  - *Read about utf-8 and unicode*

Unix Password System

- User-entered passwords are converted into bit strings.
- A salt (set by the system during user account creation and stored in the account database) is also used as input.
- A hash / ciphertext of the password (using only 64 bits of result) and a 12 bit salt are computed, the value encoded in printable ASCII characters, and stored in the account database.
  - ellen1 (user name)
  - ri (salt)
  - ri79KNd7V6.Sk (encrypted password)
- The hash algorithm varies. In early systems, the bit-string for the password is treated as a key for (a variant of) DES. The salt is used to indicate which DES variant to use (a salt all of its bits 0 results in DES being selected).
- In modern systems, the hash is based on many different modern hash function such as MD5, SHA1. Blowfish block cipher, etc.

Example: salt + hashing

User id	Salt value	Password hash
Bob	5b7	$h(5b7 \| password_{Bob})$
Alice	f3c	$h(f3c \| password_{Alice})$
Trudy	33a	$h(33a \| password_{Trudy})$

The hash function encodes the output into a printable string of characters

Note: the salt value is in the open; without salt, someone could simply hash a dictionary and compare against all hashed entries

Features of Unix passwords

- The user must know the password to authenticate himself/herself
- That creates problems for remote authentication---the password must be sent over the network connection. This requires extraneous methods to protect the connection.
- If the hashed password and salt are obtained (say, by compromising the user database), it is possible to perform brute force or dictionary attacks to recover the password.

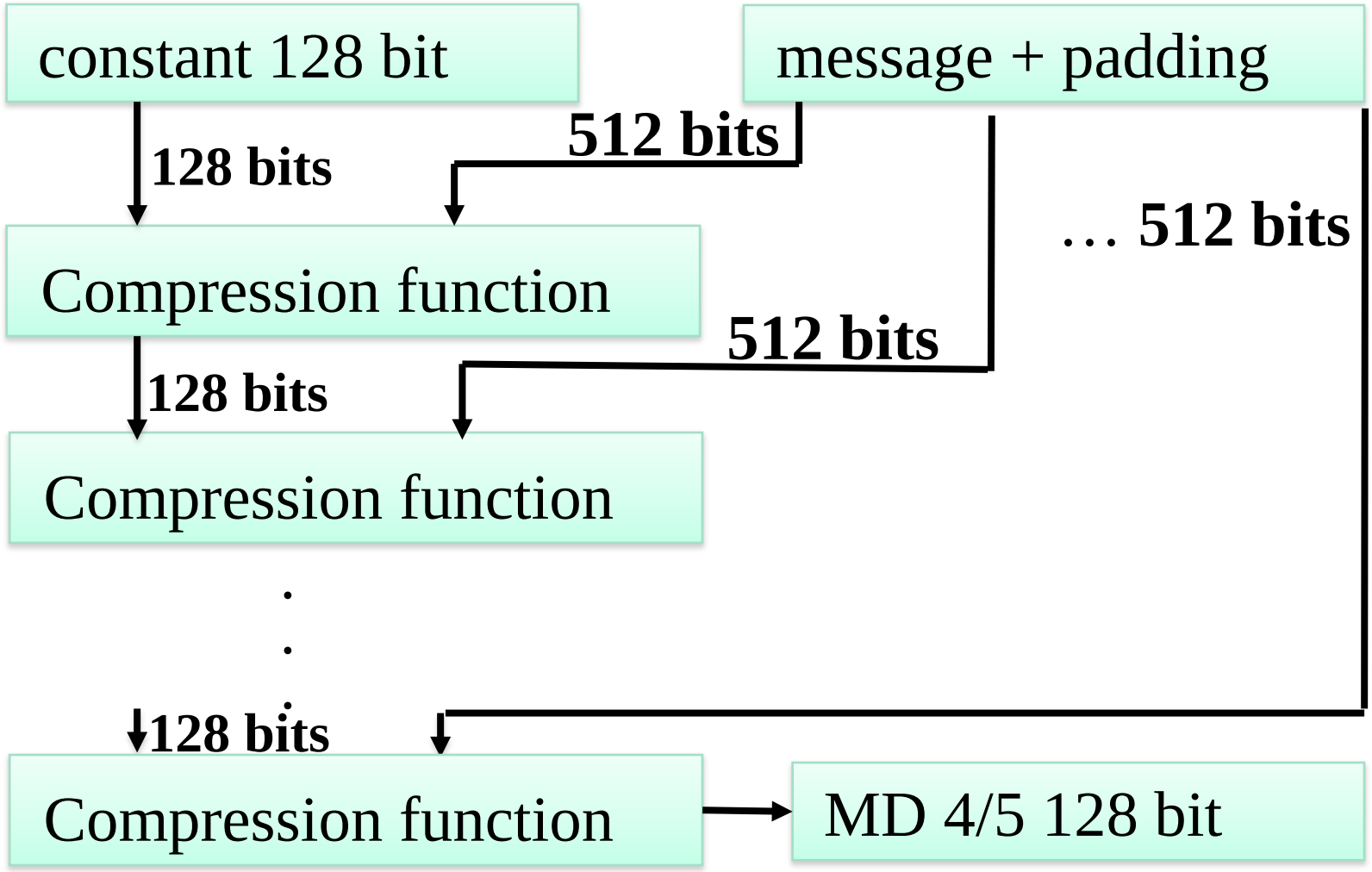
MD4, MD5

- Creates a 128 bit digest = 4 × 32 bits (4 words)
- Invented by Ron Rivest
- Main invention is the compression function:

  $\{0, 1\}^{640 \text{ bits}} \rightarrow \{0, 1\}^{128 \text{ bits}}$

- The input string is padded by a 1 followed by 0's until the length is 448 mod 512. Then the *length* of the input string is appended as a 64 bit value. The input is now a multiple of 512 bits.
- The MD construction is then used with the compression function

MD4/MD5 Pictorial View



The Compression Function

- Let the message length (in 32 bit words) be N. Then there are N/16 rounds of use of the compression function.
- A 4-word buffer (A, B, C, D) is used in each round to change the bits of this buffer using the 16 word message bits, an auxiliary table and operations such as *xor*, *not*, *and*, *or*, *rotate*. In each round the 16 word message is manipulated to create a new buffer entry.
- The last entry is the 4 word hash.
- https://tools.ietf.org/html/rfc1321

LM or LANMAN

- LM – used by Microsoft Windows prior to NT.
- User's password is a max of 14 bytes.
- Result is a 16 byte value
- LM is consider broken and can easily be inverted

The LM hash algorithm

- The user's password is converted to uppercase.
- The password is null-padded to 14 bytes.
- The modified password is split into two 7-byte halves.
- These values are used to create two DES keys (one from each 7-byte half).
- Each of these keys is used to DES-encrypt the ASCII string "KGS!@#$%“.
- The result is two 8-byte values which are concatenated to form a 16-byte value - the LM hash

NTLM or NT LAN Manager

- Successor to LM, introduced for Windows NT
- NTLM is a challenge – response authentication *protocol* to authenticate clients to servers
  - The protocol does much more than simply store a hash value to check against
- There is however also an actual stored hash associated with it
- NTLM tries to be backward compatible with LM
- It is not advised to use either versions 1 or 2 of the protocol. Instead Kerberos is recommended.

The NTLM hash algorithm

- The user's password is first changed by adding the null byte (all 0's) after each byte of the password.
- The modified password is then hashed using MD4.
- The result is a 16 byte value – the NTLM hash.

LM, NTLM Storage and Hashes

- Stored in the Windows System + SAM files
- Example entry in a SAM file (text)

xxx:1010:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::

- Structure is:

Username:RID:LMHash:NTLMHash:::

- Note that the entry starting with aad3 (after second semicolon) is the LM hash and the second entry starting with 31 after the next colon is the NTLM hash
- Can you figure out what these are hashes of?

Some example hashes of importance Supported by John the Ripper

- afs - Kerberos AFS DES
- bf - OpenBSD Blowfish
- crypt
- hmac-md5
- nt – NT MD4
- raw-md4
- raw-md5
- pix-md5
- Versions of Sha-1, Sha-256, Sha-512 etc.
- What about RAR and Truecrypt?
- Etc., …., Etc.

http://pentestmonkey.net/cheat-sheet/john-the-ripper-hash-formats

Cisco-PIX or Not?

- Cisco PIX hashes consist of a 12 byte digest encoded as a 16 character HASH64-encoded string. The algorithm is:

1. The (enable) password should be truncated to 16 bytes, or the right side NULL padded to 16 bytes, as appropriate.
2. Run the result of step 3 through MD5.
3. Discard every 4th byte of the 16-byte MD5 hash, starting with the 4th byte.
4. Encode the 12-byte result using HASH64 (assume standard Base64)

What kind of hash is the following 16 byte string? Is it a Cisco-PIX hash?

2/dBg7m+jWhQiZ==