Applied Cryptography
Spring Semester 2023
Lectures 10, 11, 12 and 13

Kenny Paterson (@kennyog), Felix Günther

Applied Cryptography Group

https://appliedcrypto.ethz.ch/

Overview of lectures

- Introducing cryptographic hash functions
- Security properties of hash functions
- Constructions for hash functions:
  - Merkle-Damgård paradigm
  - Constructions from block ciphers
  - SHA-1, SHA-2
  - SHA-3
- Application: password hashing

Introducing hash functions

Introducing hash functions

Definition

An *n*-bit (cryptographic) hash function is an efficiently computable function

$$H: \{0,1\}^* \rightarrow \{0,1\}^n$$

mapping an arbitrary length input string to a fixed-length hash value (sometimes called a message digest).

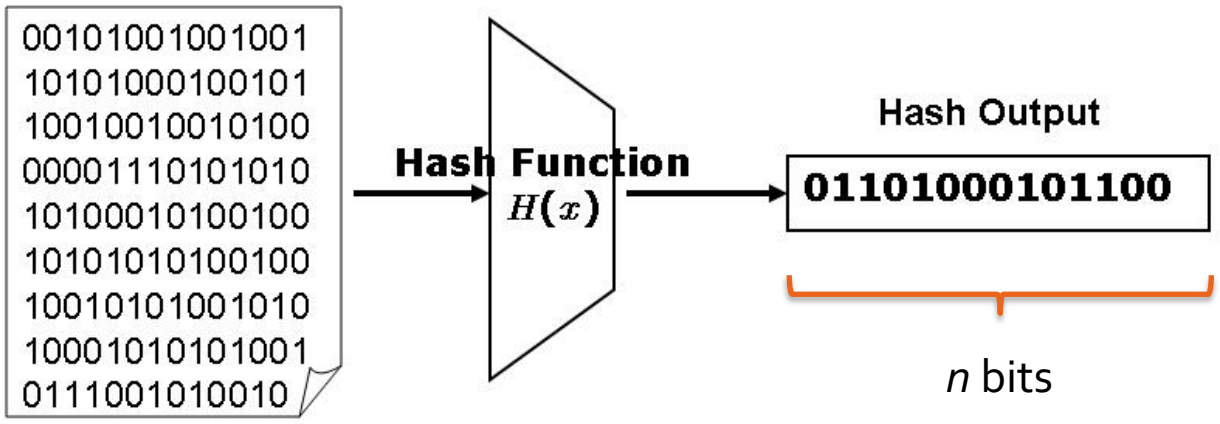
NB Hash functions are not a keyed primitive.

Random oracle model

Cryptographers often model hash functions as behaving like random functions: given any input, the output is a random *n*-bit string.

This is a strong idealization which is useful in formal security analysis (but which is not actually sound formally).

Hash functions – visualisation



Applications of hash functions

- message fingerprinting
  - file integrity checking, file sharing
- message authentication codes (e.g., HMAC)
- signature schemes (hash-then-sign paradigm)
- key derivation
  - e.g., to derive keys from 'raw data' in Diffie-Hellman key exchange
- password hashing
- commitment schemes
- pseudorandom number generators
- stream ciphers (hash function in counter mode)
- Bitcoin (proof-of-work schemes)
- post-quantum signature schemes

Standards for hash functions

NIST

SHA-1 (until 2010)

SHA-2 (includes SHA-224, SHA-256, SHA-384, SHA-512)

SHA-3 (Keccak)

NESSIE

SHA-256, SHA-384, SHA-512

WHIRLPOOL

CRYPTREC

SHA-256, SHA-384, SHA-512

Security properties of hash functions

Security goals for hash functions

Primary security goals (informal)

- pre-image resistance (one-wayness): given h, it is infeasible to find $m \in \{0, 1\}^*$ such that $H(m) = h$.
- second pre-image resistance: given m_1, it is infeasible to find $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$.
- collision resistance: it is infeasible to find collisions, i.e., a pair of messages $m_1 \neq m_2$ such that $H(m_1) = H(m_2)$.

Security goals for hash functions

Secondary security goals (informal)

- near-collision resistance: it is infeasible to find $m_1 \neq m_2$ such that $H(m_1) \approx H(m_2)$
  - (e.g. "≈" might mean hashes agree on most output bits).
- partial pre-image resistance 1: given $H(m)$, it is infeasible to recover any partial information about m.
- partial pre-image resistance 2: given a target string t of bit-length l, it is infeasible to find $m \in \{0, 1\}^*$ such that $H(m) = t \,||\, z$ in time significantly faster than 2^l hash evaluations.
  - Security of bitcoin mining depends on this latter property, with l determining difficulty level of mining a block.

Formalising Collision Resistance

Collision resistance (CR): it is infeasible to find collisions, i.e., a pair of messages $m_1 \neq m_2$ such that $H(m_1) = H(m_2)$.

- Collisions must *exist*, because the input space of the hash function is much larger than the output space.
- So for any fixed hash function H there is an efficient algorithm A that outputs collisions: a pair of colliding messages for H is is simply hard-coded into A.
- This means we can't have a security definition for CR that quantifies over *all* efficient algorithms A.
- That is, we **cannot** have a definition in the style:

> *"for all A running in time t, the probability that A outputs a collision is bounded by ε".*

Formal definition of a CR adversary

Definition: Let $H: \mathcal{X}' \to \mathcal{X}$ be a hash function.

Algorithm A is said to be a (t,ε)-CR adversary against H if A runs in time t, and with probability ε, outputs distinct $m_0, m_1 \in \mathcal{X}'$ such that:

$$H(m_0) = H(m_1).$$

We define $\mathrm{Adv}^{CR}_{H}(A) := \varepsilon$.

Notes:

- This defines the notion of a CR adversary.
- In proofs of security using CR, we will always **construct** an explicit CR adversary B from a starting adversary A against whatever scheme we are analysing.
- We will analyse the resource consumption and advantage of B in terms of that of A.
- Interpretation: if we could find an efficient A, then we could construct an efficient CR adversary B.

Relationships between security goals

Theorem (Collision resistance implies second pre-image resistance)

Any hash function that is collision-resistant is also second pre-image-resistant.

Proof (informal): Assume H is collision-resistant but not second pre-image-resistant. By the latter property one could pick any message m_1, find a second pre-image message $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$, and output hash collision (m_1, m_2), contradicting the assumed collision resistance of H.

Formalisation: see exercises.

Collision resistance and pre-image resistance

Example

Suppose G is a collision-resistant n-bit hash function.

Define a new $(n+1)$-bit hash function H as follows:

$$H(m) = 1 \,||\, m \qquad \text{if } |m| = n;$$

$$H(m) = 0 \,||\, G(m) \qquad \text{otherwise.}$$

H is still collision-resistant (why?) but, given a **random** $(n+1)$-bit challenge string t, we can expect to find a pre-image for t half of the time (why?).

- This is a "pathological" (i.e. artificial) example.
- Its purpose is to show that we cannot expect to find a general proof that CR implies pre-image resistance under a **specific** definition for the latter.
- The definition here involves sampling a random challenge value y from the **range** of the hash function – here $\{0,1\}^{n+1}$.

Collision resistance and pre-image resistance

Different formalisation of pre-image resistance:

1. Challenger samples a value x uniformly at random from the **domain** of the hash function, sets $y=H(x)$ and gives y to adversary A.
2. A runs and returns a value x'.
3. A wins if $y=H(x')$.

- With this definition, we **can** prove that collision resistance implies pre-image resistance, provided the domain is (much) larger than the range.
- Definitions matter!
- Further discussion in exercises.

Generic attacks on hash functions

Generic attack on:

- Pre-image resistance: given $h \in \{0, 1\}^n$, if H behaves like a random function, then by iterating over about 2^n different messages m and for each one evaluating $H(m)$, we should find m such that $H(m) = h$.
- Second pre-image resistance: given m_1, if H behaves like a random function, then by iterating over about 2^n different messages m_2 and for each one evaluating $H(m_2)$, we should find m_2 such that $H(m_1) = H(m_2)$.

Designer's goal: ensure there are no better attacks on pre-image resistance and second pre-image resistance than these generic attacks.

(NB n, the output size, needs to be big enough to make these attacks infeasible.)

Generic attacks on hash functions (2)

Generic attack on:

- Collision resistance: if H behaves like a random function, then by iterating over about $\mathbf{2}^{n/2}$ different messages m and for each one evaluating $H(m)$, we should find m_1 and m_2 such that $H(m_1) = H(m_2)$.

Analysis relies on the birthday theorem: if elements are drawn at random from a set of size s, then after $\sqrt{s}$ trials, we expect a collision in the sampled elements with about 39% probability.

Designer's goal: ensure there are no better attacks on collision resistance than this generic attack.

Birthday attack probability analysis

Suppose we have a set of s objects, and we sample at random from the set t times.

Probability that all the samples are different is:

$$1 \cdot (s-1)/s \cdot (s-2)/s \cdot \ldots \cdot (s-t+1)/s$$

Hence probability of at least one collision, i.e. at least one repeated sample, is:

$$1 - (1-1/s)(1-2/s)\ldots(1-(t-1)/s)$$

$$= 1 - \exp(\Sigma_j \log(1 - j/s))$$ (use $\exp(\log(x)) = x$ and $\log(\text{product}) = \text{sum}(\log)$)

$$\approx 1 - \exp(-\Sigma_j\ j/s)$$ (use $\log(1-x) \approx -x$ for small x)

$$= 1 - \exp(-t(t-1)/2s)$$ (sum of integers $1,\ldots t-1$ is $t(t-1)/2$)

$$\approx 1 - \exp(-t^2/2s)$$

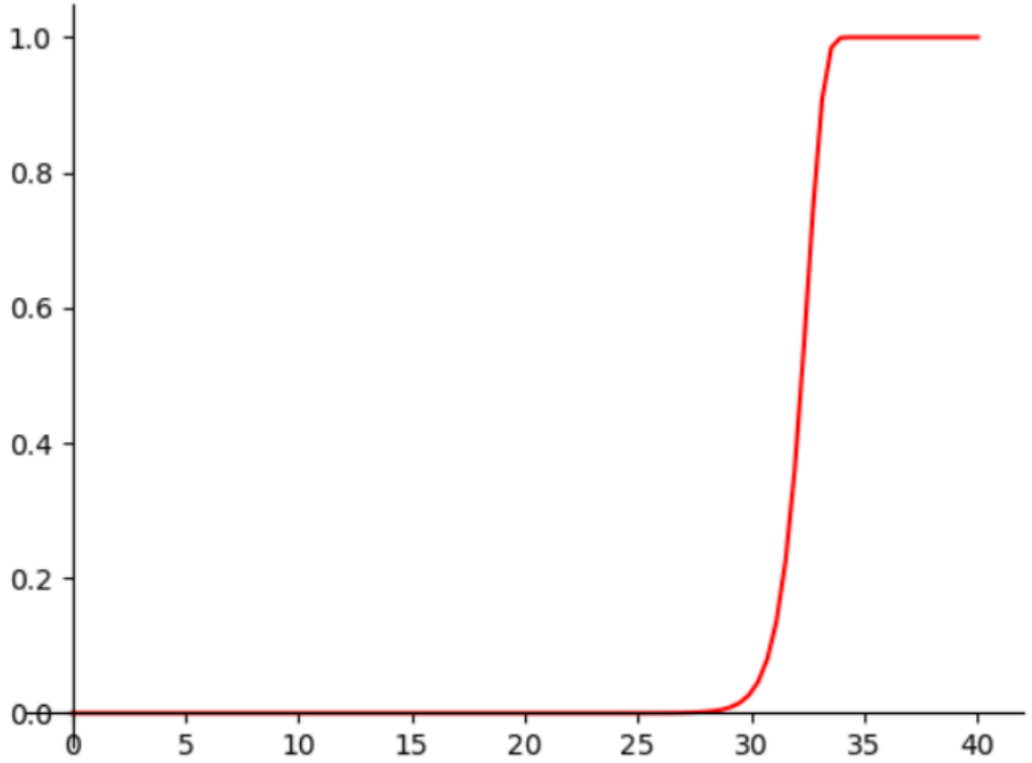
For $t \approx s^{1/2}$, this probability is about 0.39.

For $t \approx 1.17 \cdot s^{1/2}$, this probability is about 0.5.

For $t \approx 3.03 \cdot s^{1/2}$, this probability is about 0.99.

The "birthday paradox":
For n = 365, we get a
≈ 50% probability
already for t = 23.

Birthday attack visualisation



Collision probability for $s = 2^{64}$, $\log_2$ scale on x-axis.

Birthday attack probability analysis

Conclusions from the analysis:

- An n-bit hash function offers only $n/2$ bits of security* against the generic collision attack.
- So we need hash functions with 256-bit outputs to achieve 128-bit security level, etc.
- SHA-1, still in widespread use, has an output size of 160 bits, so can only reach 80-bit security level against generic collision attack.

*Meaning attack cost is $2^{n/2}$ "operations" + memory in some cost model.

Constructions for hash functions

Normal hash functions are not good (for crypto)!

Hash functions are widely used in computer science for non-cryptographic purposes, especially in connection with probabilistic data structures.

- Hash tables
- Caching strategies
- Bloom filters, Cuckoo filters and other probabilistic data structures.

Non-cryptographic hash functions are weak!

Hash functions like CRC, Fletcher, Adler, Zobrist, Jenkins, MurmurHash, CityHash are absolutely useless for any cryptographic purposes.

Example: Insecure hashing in WEP

The WLAN encryption protocol WEP is broken in part because CRC was used as a hash function in an attempt to provide message integrity (problem: CRC is linear in the bits of the message).

Hash function designs

Several design paradigms:

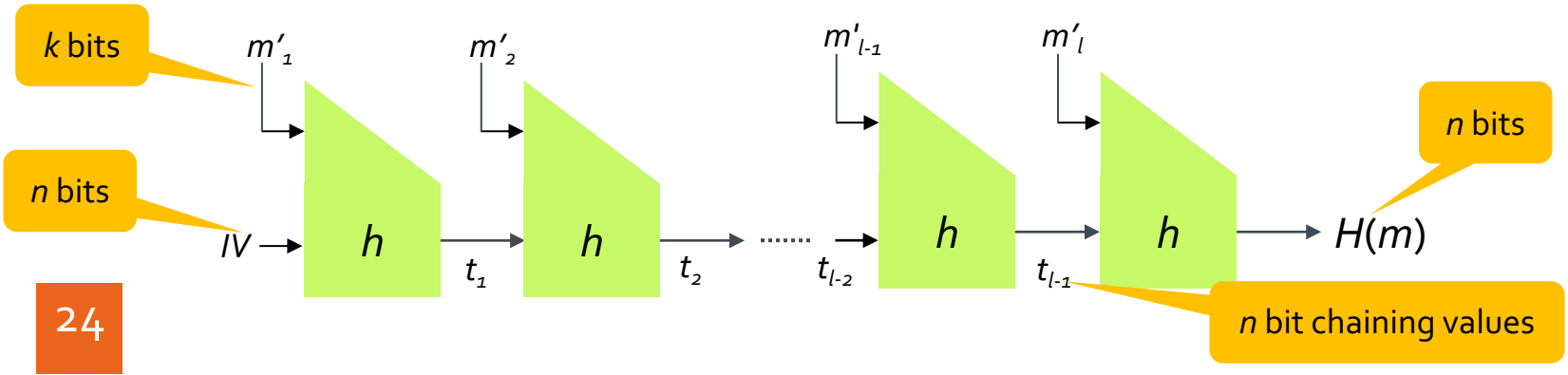
- Based on block ciphers.
- Dedicated designs.

- One difficulty in designing a hash function is that it must be able to process arbitrary length messages.
  - Leads naturally to the idea of an iterative design

Merkle-Damgård iterated hashing

Construct a hash function from a compression function.

- Let k be block length, n be output length, $IV \in \{0,1\}^n$ be constant.
- Assume we have a compression function h: $\{0,1\}^{k+n} \rightarrow \{0,1\}^n$ (or h: $\{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$).
- Assume we have an injective padding scheme pad(.) such that length of m' = pad(m) is a multiple of k in bits.
- Break-up m' = pad(m) into blocks $m_1', \ldots, m_l'$ each of length k.
- Process blocks as shown **below** to compute hash value $H(m)$.
- Each step processes a chunk of k input bits; the final result is n bits in size.
- Prominent instantiations: MD5, SHA-1, SHA-2, WHIRLPOOL.



Merkle-Damgård iterated hashing (2a)

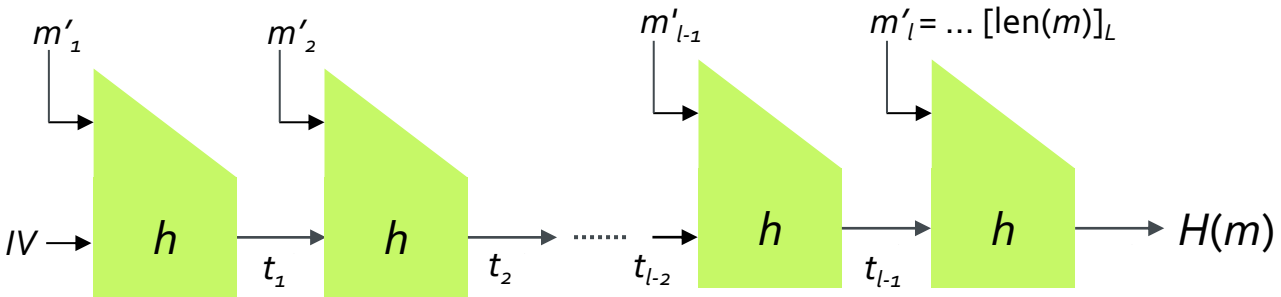
Theorem (Security of the Merkle-Damgård construction)

Suppose pad(m) transforms m into

$$m' = m \,||\, 10^t \,||\, [\text{len}(m)]_L$$

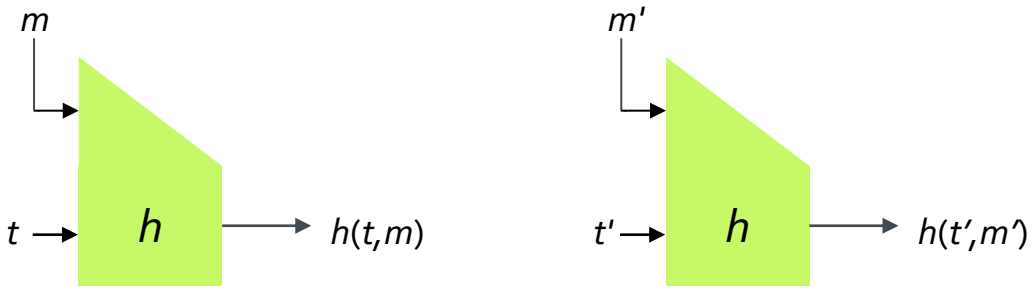
where $0 \le t < k$ is minimal such that k divides len(m') and "$[\ldots]_L$" denotes the L-bit representation of a number, where $L \le k$.

If the compression function h is collision-resistant, then so is H.



Merkle-Damgård iterated hashing (2b)

- Note that a collision in *h*, the compression function, is defined to be any pair of inputs (*t*,*m*), (*t'*,*m'*) such that *h(t,m) = h(t',m') and* (*t* ≠ *t'*) OR (*m* ≠ *m'*).
- If we think of the input to *h* as being in $\{0,1\}^{n+k}$, instead of $\{0,1\}^n \times \{0,1\}^k$, then we get a natural definition of collisions: two different (*n*+*k*)-bit inputs *t* || *m* and *t'* || *m'* producing the same outputs.



Merkle-Damgård iterated hashing (3)

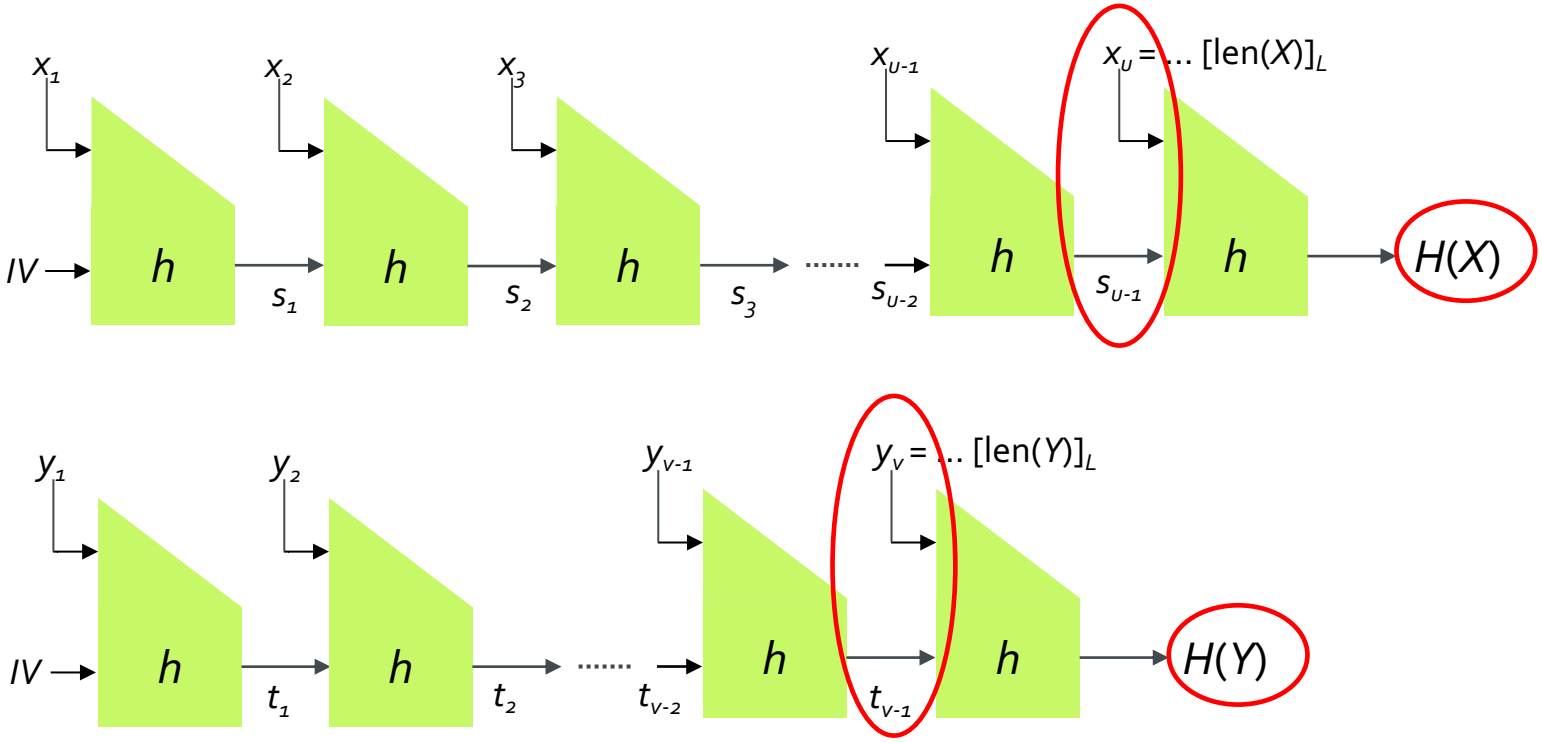
Proof sketch (of security of the Merkle-Damgård construction)

- Let *A* be an adversary against CR of hash function *H* built from compression function *h* using the MD construction. We construct from *A* an adversary *B* that breaks CR security of *h*.
- Suppose *A* successfully outputs a colliding pair $X \neq Y$.
- Note that pad(*X*) and pad(*Y*) need not have the same number of blocks.
- Let their blocks after being padded be x_i, y_j.
- Write pad(*X*) = $x_1, x_2, \ldots, x_u$ and pad(*Y*) = $y_1, y_2, \ldots, y_v$.
- Let s_i be the chaining values for *X* and t_i be the chaining values for *Y*.
- So, looking at last blocks in the two chains, we have

$$h(s_{u-1}, x_u) = H(X) = H(Y) = h(t_{v-1}, y_v).$$

- **Case 1: $(s_{u-1}, x_u) \neq (t_{v-1}, y_v)$**
  If $(s_{u-1}, x_u) \neq (t_{v-1}, y_v)$ then the pair of inputs (s_{u-1}, x_u) and (t_{v-1}, y_v) is an *h*-collision.
- *B* outputs this collision and terminates.

Merkle-Damgård iterated hashing (3a)



- By assumption: $h(s_{U-1},x_U) = H(X) = H(Y) = h(t_{V-1},y_V)$.
- If $(s_{U-1},x_U) \neq (t_{V-1},y_V)$: we have a collision in compression function *h*.

Merkle-Damgård iterated hashing (4)

Proof sketch (of security of the Merkle-Damgård construction)

- **Case 2: $(s_{U-1}, x_U) = (t_{V-1}, y_V)$**
  Now we consider the other case where $(s_{U-1}, x_U) = (t_{V-1}, y_V)$.
- Since x_U, y_V both uniquely encode the lengths of X, Y respectively, we can deduce from $x_U = y_V$ that in fact $u = v$ and the messages are of identical length.
- Now since $s_{U-1} = t_{U-1}$, for the previous inputs to the second-to-last blocks we have:

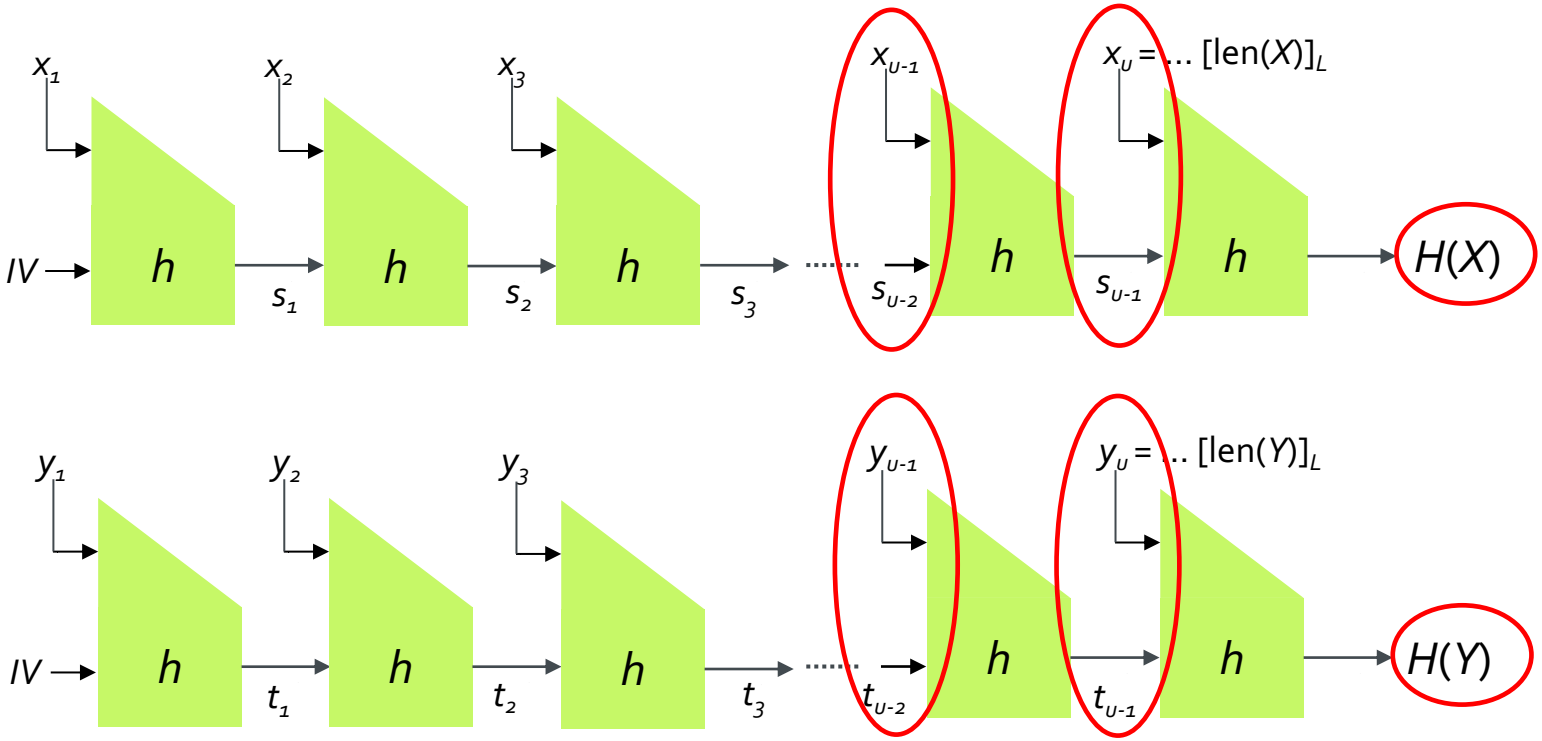
$$h(s_{U-2}, x_{U-1}) = s_{U-1} = t_{U-1} = h(t_{U-2}, y_{U-1}).$$

- Apply the same argument as before: if $(s_{U-2}, x_{U-1}) \neq (t_{U-2}, y_{U-1})$, then the pair of inputs (s_{U-2}, x_{U-1}) and (t_{U-2}, y_{U-1}) is an h-collision; B outputs this collision and terminates.
- Otherwise, $(s_{U-2}, x_{U-1}) = (t_{U-2}, y_{U-1})$, and from $s_{U-2} = t_{U-2}$, we must have same outputs under h for the previous pairs of inputs, i.e.:

$$h(s_{U-3}, x_{U-2}) = s_{U-2} = t_{U-2} = h(t_{U-3}, y_{U-2}).$$

- Keep going!
- The process must end with a collision in h, otherwise we would eventually find that all blocks of pad(X) equal those of pad(Y), contradicting $X \neq Y$.

Merkle-Damgård iterated hashing (4a)



$(s_{U-1}, x_U) = (t_{V-1}, y_V)$ implies:

1. $\text{len}(X) = \text{len}(Y)$ and $u = v$.
2. $s_{U-1} = t_{U-1}$, so we have a collision in h one step earlier.

Now apply same argument to inputs (s_{U-2}, x_{U-1}) and (t_{U-2}, y_{U-1}).

Merkle-Damgård iterated hashing (5)

Proof sketch (of security of the Merkle-Damgård construction)

- This completes the proof.
- Note that the running time of *B* is essentially that of *A.*
- Moreover, if *A* is successful with some probability ε, then so is *B.*

Length encoding

- Usually the encoding of len(*m*) is shorter than a full message block of *k* bits; 64-bit encodings are typical and used in SHA-1, SHA-2.
- Fixed-length encoding implies there is a maximum message length that can be hashed: 2^L-1 bits.

Length extension attack on Merkle-Damgård

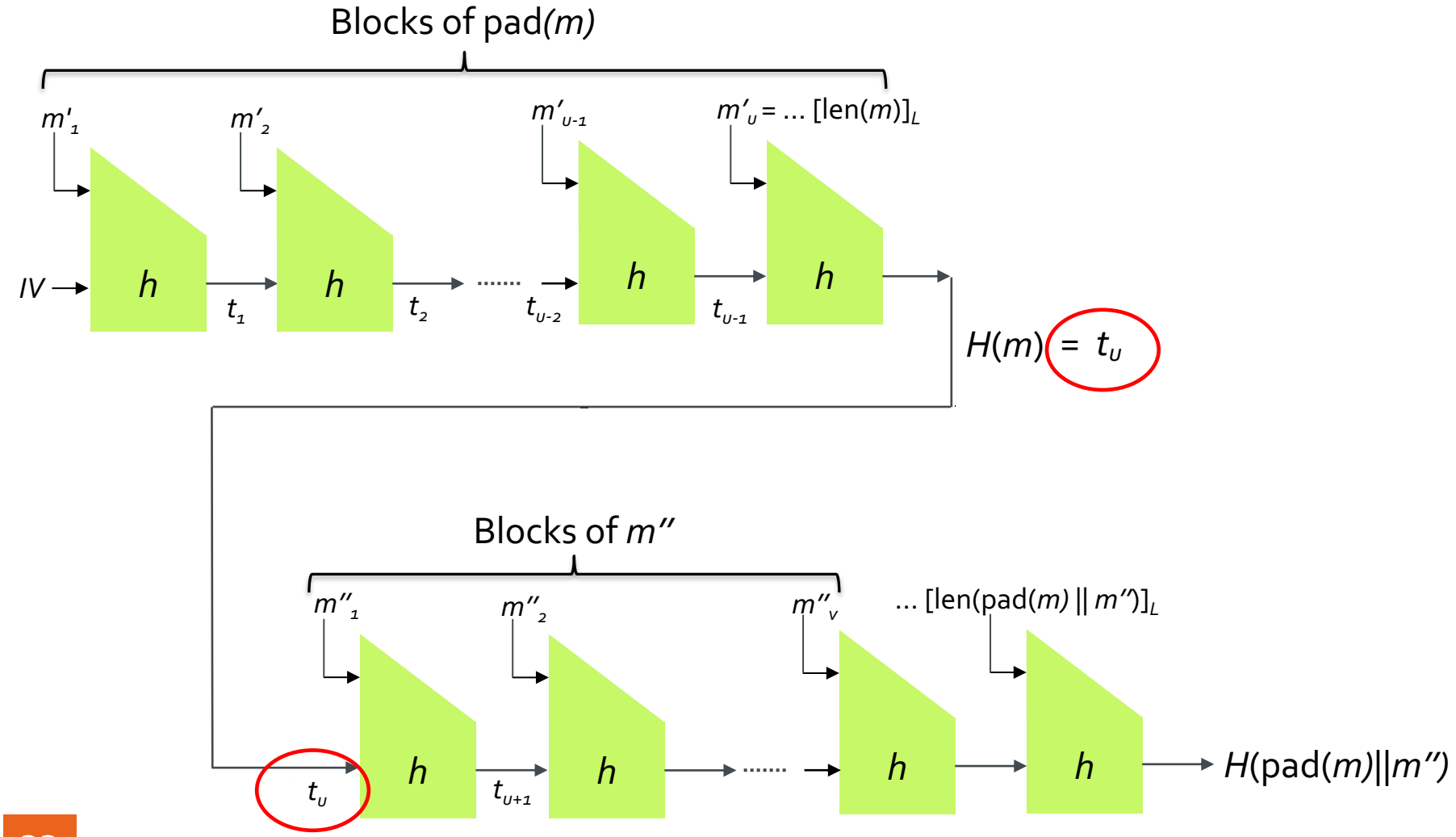
Merkle-Damgård hashing has an unfortunate length-extension property: If we know $y = H(m)$, then we can compute

$$y' = H(\text{pad}(m) \,||\, m'')$$

for any m'', even without knowing m.

- Note that the input here "pad(m) || m'' is an extension of pad(m).
- To see why this works, note that hashing m involves iterating the compression function on pad(m), with the hash output being the last output; this becomes an intermediate chaining value when computing the hash of pad(m) || m''. **(See diagram on next slide.)**
- Problematic for certain applications, e.g. MAC or key derivation function constructed from the hash function by concatenating key and message.

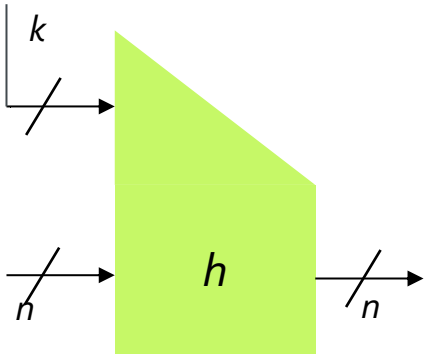
Length extension attack on Merkle-Damgård (2)



Constructing compression functions from block ciphers

We've reduced our problem of constructing a good hash function to that of constructing a good compression function *h*.

- Interface requirement: h:$\{0,1\}^{n+k} \longrightarrow \{0, 1\}^{n}$, that is *h* maps *n*+*k* bits to *n* bits.
- Security requirements: collision resistant, pre-image resistant,…

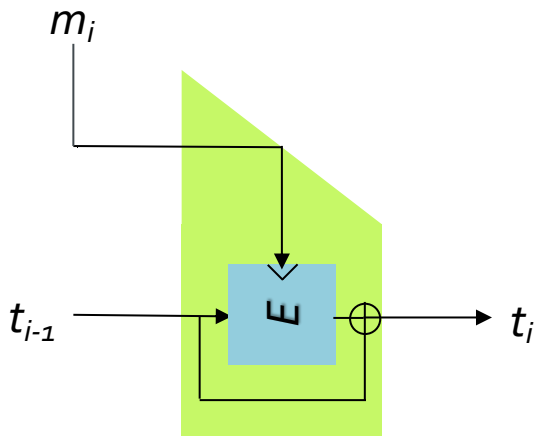


Can we re-use existing components, like block ciphers?

Constructing compression functions from block ciphers

Constructions from block ciphers.

- Davies-Meyer, Matyas-Meyer-Oseas, Miyaguchi-Preneel.
- Some need additional key processing to map input to key space.
- Main issue is that hash output size = block cipher block size, so need large block size *n* to avoid $2^{n/2}$ birthday attack.
- E.g., SHA-2 uses n = 256 and k = 512
- Secondary issue is that, e.g. in the Davies-Meyer construction, message blocks are used to set key, so rekeying of block cipher needs to be fast.
- One can show that if *E* is an ideal cipher, then Davies-Meyer gives a collision-resistant compression function.



$t_i = h(t_{i-1}, m_i) = E(m_i, t_{i-1}) \oplus t_{i-1}$

Davies-Meyer:
chaining variable becomes message input; message input becomes key.

Second pre-image resistance of Merkle-Damgård

Merkle-Damgård hashing with Davies-Meyer construction of *h* does not achieve desired generic security against second pre-image attacks!

- Recall that the desired security level is 2^n hash operations to find a second pre-image (i.e., $m_2 \neq m_1$ such that $H(m_1) = H(m_2)$, when given m_1)
- It is possible to find second pre-images in $O(2^{n/2})$ hash computations, but the known attack uses very long messages.
- Attack focusses on finding fixed points for *h*: values *t* such that $h(t,m) = t$.

$$h(t,m) = t \Leftrightarrow E(m,t) \oplus t = t \Leftrightarrow E(m,t) = 0^n \Leftrightarrow t = E^{-1}(m,0^n).$$

- Hence for any input block *m*, we can find a *t* for which *t* is a fixed point of *h*.
- Details of the full attack are omitted.

Dedicated Hash Functions: Examples

MD5 (replacing MD4)

128-bit hash , designed by Ron Rivest in **1991**, specified in RFC 1321.

SHA-1 (replacing SHA-0)

Tweak of SHA-0, 160-bit hash, designed by NSA, standardised by NIST in FIPS PUB 180-1 (**1995**).

Used extensively in TLS/SSL, PGP, SSH, S/MIME, IPSec

SHA-2 family (**SHA224**, **SHA256**, **SHA384**, **SHA512**)

Designed by NSA, standardised by NIST in FIPS PUB 180-2 (**2002**), recommended for US government use.

SHA-256 very commonly used, other options less so (output size, speed).

SHA-3

Public design, standardised by NIST in FIPS PUB 202 (**2015**), complementary to SHA-2 family (not a replacement), fundamentally different construction.

Whirlpool

512-bit hash (based on modified AES).

Used in TrueCrypt (open-source encryption toolkit).

(**RIPEMD**) **RIPEMD-160, BLAKE, BLAKE2…**

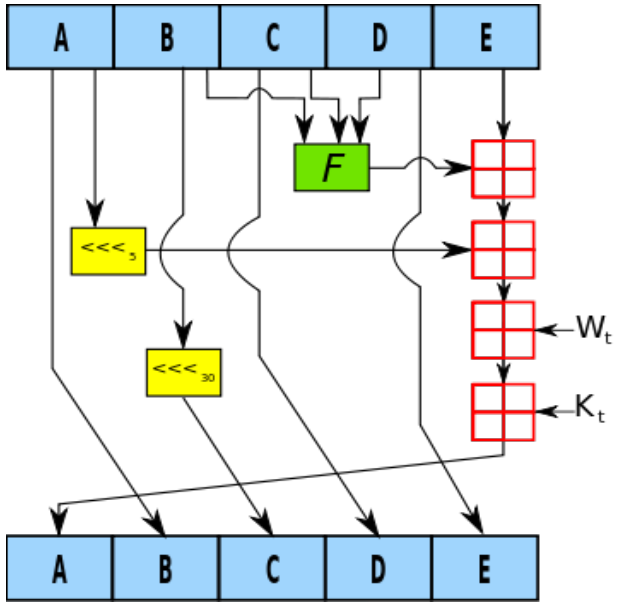
Dedicated Hash Functions: the MDx-family

MDx-family:

- Derived from an early design from Ron Rivest in the early 1990s.
- Very efficient and particularly suited to software implementation on 32-bit machines.
- Further motivation for custom-made hash functions: ban on export of cryptographic algorithms in the 1980s and 1990s (no ban for hash functions).
- MD4, MD5, SHA-1, SHA-2.
- NIST and ISO/IEC 10118-3 standards.
- MD4, MD5, SHA-1 are all broken (very badly in the case of MD4 and MD5).
- Despite wide range of choices and security issues, SHA-1 still remains widely used today, e.g. IPSec, SSL/TLS, SSH.

SHA-1

- Merkle-Damgård hash with output size *n* equal to 160 (20-byte output).
- Aimed to provide 2^{80} security against collision attacks.
- Compression function uses (implicitly) a special block cipher ("SHACAL-1") in Davis-Meyer mode, $k = 512$, $n = 160$.
- Block cipher has a round function operating on 5 x 32-bit words (chaining value) and key word W_t (message).
- Round function is iterated 80 times.
- Feistel structure. Uses mod 2^{32} additions, bit shifts, other simple bit operations (*F*).



"SHA-1". Licensed under CC BY-SA 2.5 via Commons – https://commons.wikimedia.org/wiki/File:SHA-1.svg#/media/File:SHA-1.svg

Compressed timeline of SHA-1 security story

1995: Initial analysis of SHA-0 led to a tweak to its design to produce SHA-1.

2005: First attacks on SHA-1 (Wang *et al.*): estimated cost of 2^{69} compression functions evaluations (not implemented); expert commentators: it's time to start moving away!

2005 – 2015: Lots more analysis, collisions estimated in 2^{61} compression function evaluations; no attacks on full hash function or compression function implemented, but collisions for up to 77 rounds out of 80 in compression fn.

2012: NIST starts to retire SHA-1, but then (2015) relaxes policy see https://csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions

Late 2014: Google and others start to penalize sites using SHA-1 certificates.

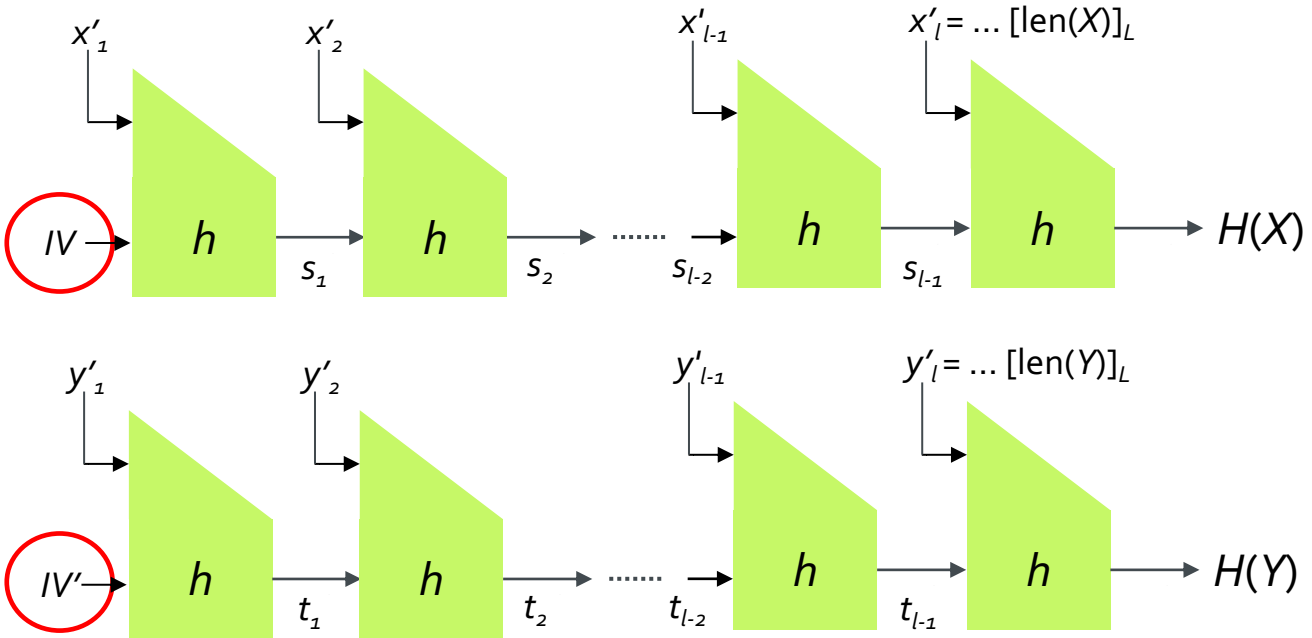
October 2015: first **freestart** collision attack announced on full SHA-1 (see https://sites.google.com/site/itstheshappening/).

February 2017: **concrete collision** on full SHA-1 – see https://shattered.it/.

January 2020: **chosen-prefix** collision on full SHA-1 – see https://eprint.iacr.org/2020/014 (consequences for certificate forgeries).

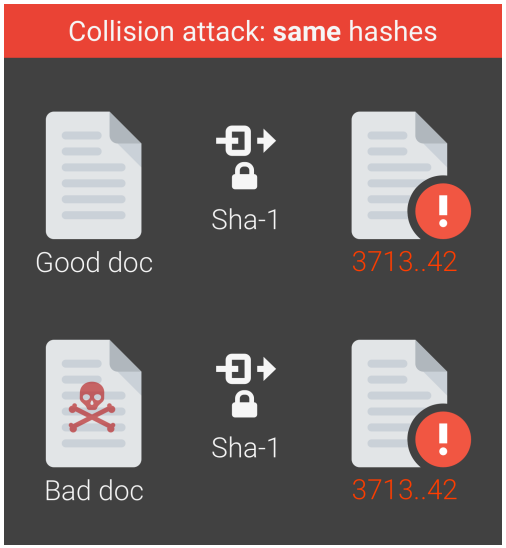
The SHAppening (Stevens, Karpman, Peyrin)

- **Freestart** collision is significantly weaker attack than full collision.
- Attacker gets to choose two different IVs in Merkle-Damgård construction.
  - In reality, the IVs are fixed to a single value!
- Attack used 64 GPUs and took a few days.



The SHATTERED attack

- **Full collision** on SHA-1 by Stevens & Karpman from CWI, Bursztein, Albertini & Markov from Google, Feb. 2017.
- Builds on freestart collision by Stevens, Karpman & Peyrin, 2015.
- Required 2^{63} SHA-1 computations, ≈ 6500 CPU-years + 100 GPU-years.
- Allowed creation of pdf documents having colliding hash values.
- **Estimated cost of full collision attack on SHA-1: 100k-500k USD.**



from https://shattered.it

SHA-1 is a Shambles: Chosen-prefix collisions on SHA-1

- **Chosen-prefix and full collisions** on SHA-1 by Leurent & Peyrin, 2020.
- Required $2^{61.2}$ SHA-1 computations for full collisions.
- Required $2^{63.4}$ SHA-1 computations for chosen prefix collisions: given p_1, p_2, find m_1, m_2 such that:

$$H(p_1 \,||\, m_1) = H(p_2 \,||\, m_2).$$

- Attack used 900 Nvidia GTX 1060 GPUs for 2 months.
- Estimated costs: US$ 11k for a collision and US$ 45k for a chosen-prefix collision on cheaper, rented GPUs.
- Authors applied the attack to create colliding PGP identity certificates, enabling impersonation attacks against PGP users.

Phasing out broken algorithms

Question: Why does it take so long to phase out a broken cryptographic algorithm?

Possible answers:

- Once deployed, hard to phase out, especially in systems without cryptographic agility.
- Desire to maintain backwards compatibility with older software.
- Practitioners seem to require practical demonstration of attack (they are less conservative than cryptographers; this situation seems to be slowly improving).
- Lack of understanding of how attacks evolve outside narrow circle of crypto experts (attacks really do only get stronger!).
- Attacks only on collision-resistance, do not break all security properties (e.g. one wayness), so MD5 maybe still OK in some applications, leading to confusion about algorithm status.
  - Example: best pre-image attack on MD5 still requires $2^{123.4}$ effort.

SHA-2 family

- SHA-256: similar in design to SHA-1, but now using 256-bit block cipher (SHACAL-2) in Davies-Meyer mode to build compression function.
  - So n = **256**, k = 512.
  - 64 rounds, operating on 8 x 32-bit words using simple operations.
  - About twice as slow as SHA-1 for long messages (see https://en.wikipedia.org/wiki/SHA-3#Comparison_of_SHA_functions).
  - Lots of cryptanalysis since introduction in 2001, but no attack for collision finding faster than generic birthday attack (2^{128}) is currently known for SHA-256.
- SHA-224, SHA-384 and SHA-512 options also available (but less widely deployed/used).

SHA-3

- In 2007, NIST launched an AES-style competition to design new hash functions, serving as alternative to SHA-2.
- The closure of the call period was at the end of 2008.
- 63 submissions were received, 51 were selected for first round.
- Two conferences to discuss candidates were held.
- 14 semi-finalists were selected for further analysis.
- 5 finalists were selected at the end of 2010.
- Keccak* was announced as winner in 2012.
- NIST standard finally published in 2015 (FIPS 202).
- **An 8-year process in total.**

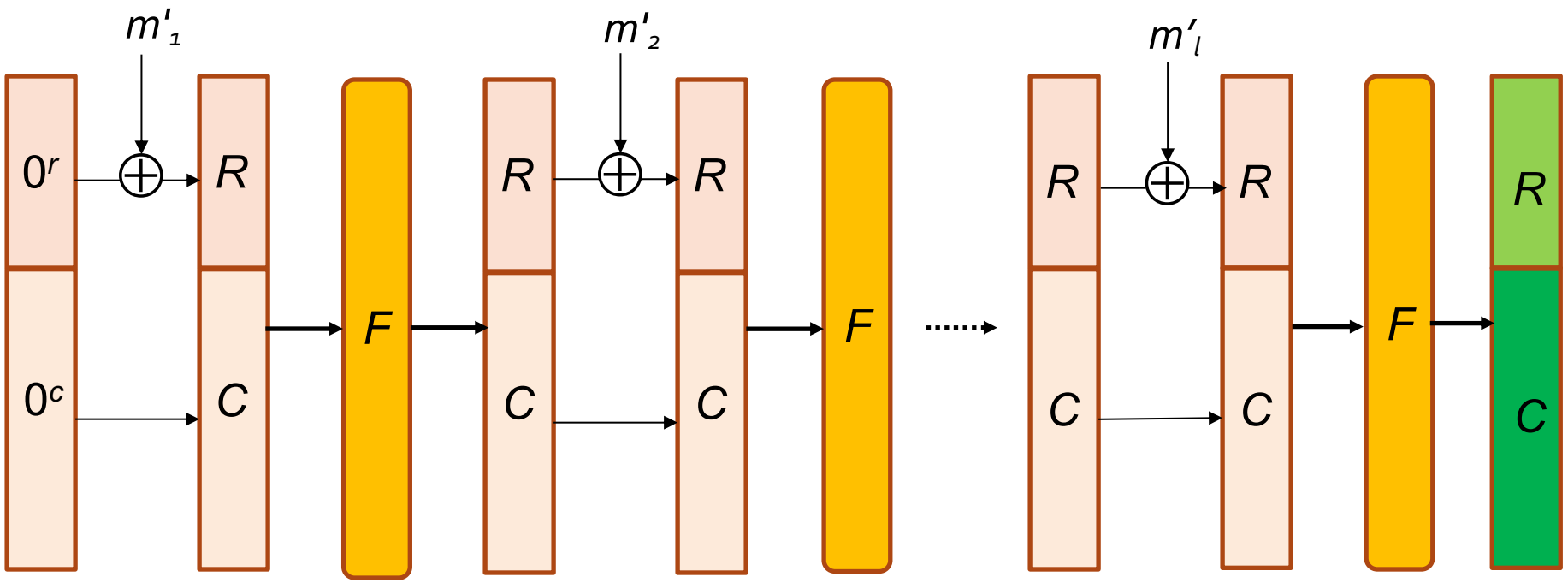
*pronounced "Ketchak"

SHA-3

- Keccak was designed by industry-based researchers from STMicroelectronics and NXP, including Joan Daemen (AES co-designer).
- Based on a new design approach (a "sponge" construction), unrelated to MD4, MD5, SHA-1, SHA-2 families / the Merkle-Damgård construction.
- Uses a "giant unkeyed permutation on bit strings" as a key element in the construction.
- Permits variable length output, very useful in applications like key derivation: eXtendable Output Function (XOF), called SHAKE.
- Avoids length extension problems of MD construction.
- Has variable throughput, allowing efficiency/security trade-offs.
- Chosen for elegance, security margins, performance and flexibility.
- Intended to complement, not replace, SHA-2: a back-up algorithm.
- SHA3-256 a bit (10-15%) slower than SHA-256.

Sponge construction – absorbing phase

Pad(m) = $m'_1\, m'_2 \ldots m'_l$ (SHA-3: pad with 10*1)



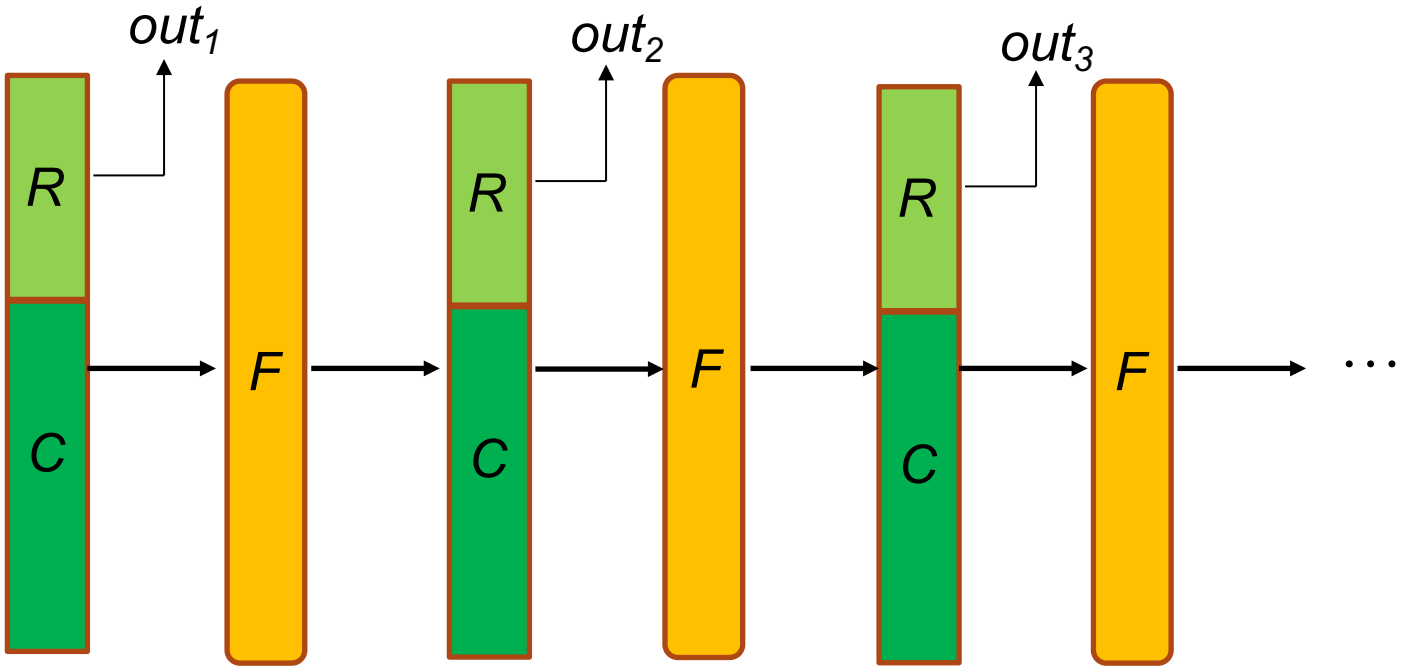
m'_i: padded message blocks, 1088 *bits* for SHA3-256.

R: outer state, r = 1088 bits for SHA3-256.

C: inner state, c = 512 bits for SHA3-256.

F: bit permutation mapping $\{0,1\}^{1600}$ to $\{0,1\}^{1600}$ ($r+c$ = 1600).

Sponge construction – squeezing phase



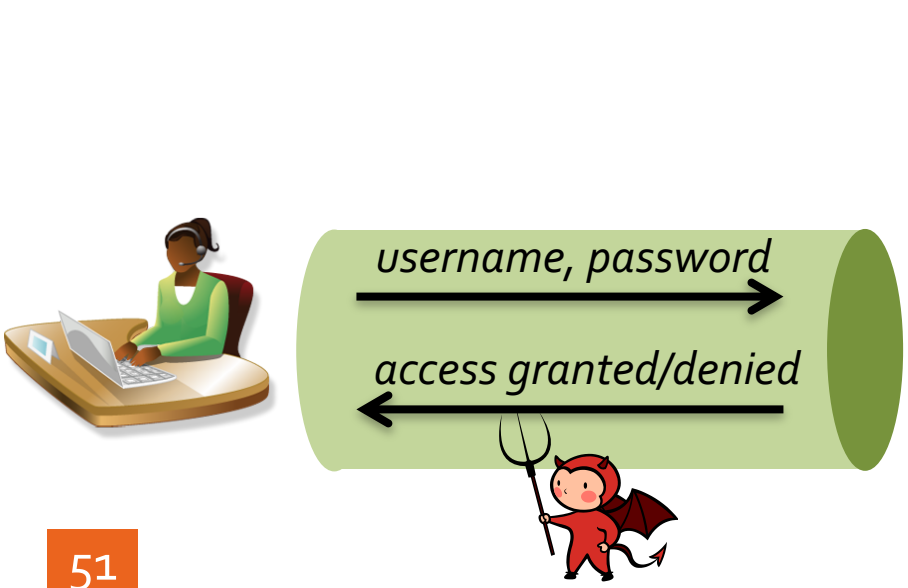
SHA-3(m) output is out_1 || out_2 || out_3 …

- Important: we don't output the entire state, only the outer state R.
- Efficiency/security trade-off intuition: the larger c, the more security, the larger r, the more throughput

Application: Password hashing

Password hashing

- Setting: users want to remotely authenticate to server using password.
- In naïve implementation: server has database with all passwords **in clear**.
- Security threats?



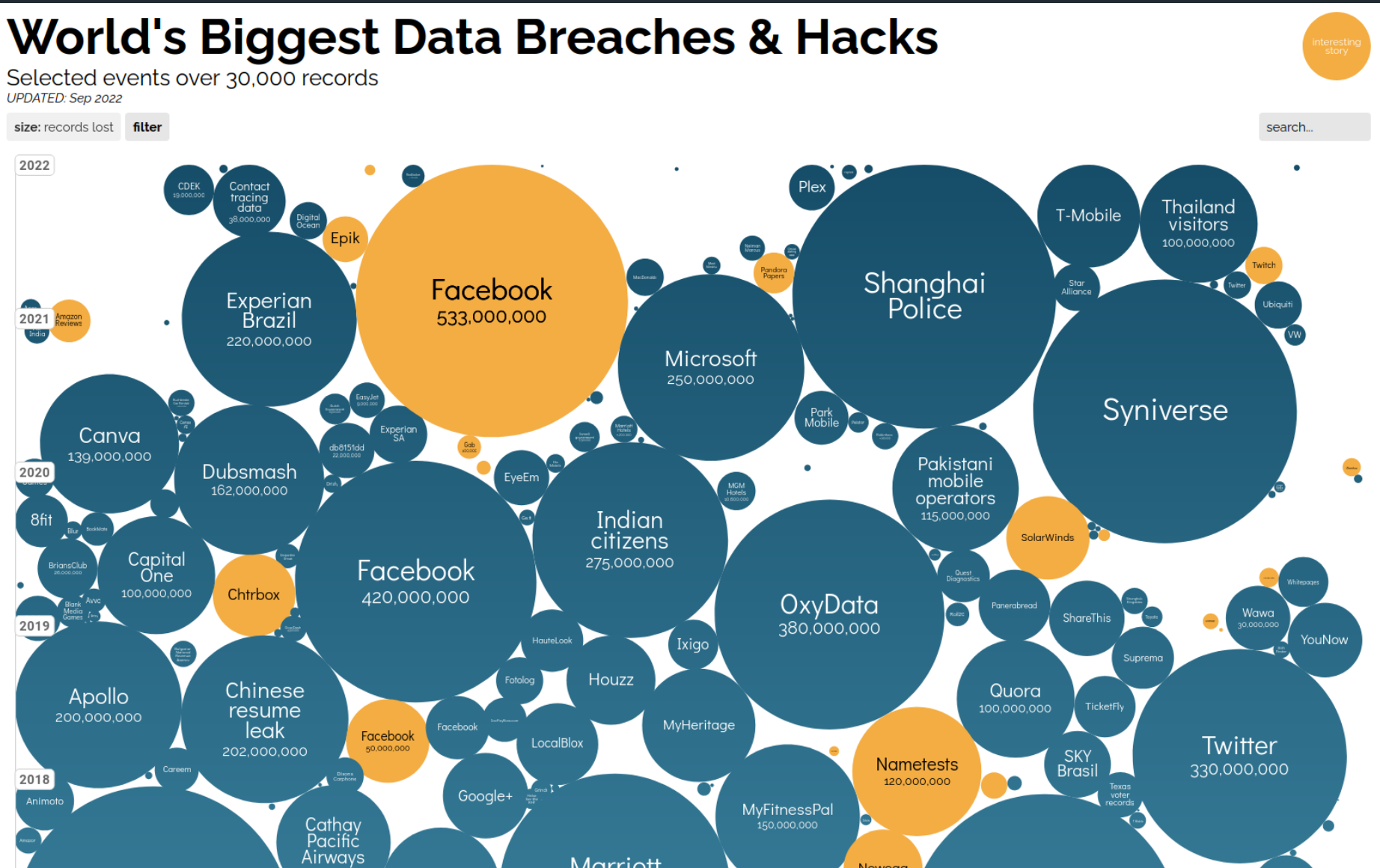
Username	Password
Alice	password
Bob	123456
…	…

Password hashing

Threat: attacker breaks into the server and steals this database and then goes on to impersonate users.

- This is a very realistic threat.
- Example: RockYou, Adobe, Experian,…

https://www.informationisbeautiful.net



Password hashing

Threat: attacker breaks into the server and steals this database and then goes on to impersonate users.

- This is a very realistic threat.
- Example: RockYou, Adobe, Experian,…

Solution: store only password hashes in database.

- For each user, store *H*(password) instead of password.
- Server now hashes received password and compares to table.
- Intuition: attacker has to reverse the hash (find pre-image) to recover the password.

Password hashing

Username	Password hashes
Alice	AB1134F3DF
Bob	A76642E3CD
…	…

Password hashing

Discussion: to what extent does this approach improve security?

Some discussion points:

- Security depends on one-wayness of hash function *H* used: if *H* can be reversed, then passwords can be found from hashes.
- Use of password crackers: build dictionary, try to hash all passwords in dictionary, check for matches.
- Sophistication of password crackers: tuning, site-specific, number-letter substitutions, rainbow tables, etc.
- Use of GPUs and special-purpose hardware to speed-up password cracking.
- Use of common passwords: makes cracking many accounts easy.

Common passwords in RockYou breach

Rank	Password	Absolute Frequency	Relative Frequency
1	123456	0.008917	0.034250
2	12345	0.002425	0.059588
3	123456789	0.002355	0.01944
4	password	0.001824	0.009130
5	iloveyou	0.001532	0.007669
6	princess	0.001021	0.005111
7	1234567	0.000666	0.003455
8	rockyou	0.000641	0.003324
9	12345678	0.000630	0.003156
10	abc123	0.000511	0.001961

TABLE I: Frequencies of RockYou top 10 passwords.

- Social app site, 2009 breach: 32 million accounts, 14 million unique passwords
- Absolute frequency: fraction amongst all accounts.
- Relative frequency: fraction amongst passwords of the same length.
- Note presence of site-specific passwords, e.g "rockyou" > 0.3%

Password hashing

- Salting: database stores

  (`salt`, H(`salt` || `password`))

  where salt is an account-specific random value.
- Effect on security?

- Iteration of hash function: PBKDF
- Effect on security?

- Memory-hard hash functions: bcrypt, scrypt.
- Effect on security?

Salting

- Add a random, account-specific value, called a *salt*, in each hash calculation.
- This means that each account has to be attacked individually (via password hashing).
- Store the salt values as a field in the database of usernames and password hashes.
- 64 bits of salt is typical: enough to prevent collisions in random choices of salt values (birthday bound again) and to devalue pre-computation by an attacker.

Username	Salt	Password hashes
Alice	B62477A8	AB1134F3DF
Bob	C2EF1377	A76642E3CD
…	…	…

$H(\texttt{salt} \,\|\, \texttt{password})$

Hash iteration

- Slow down password cracking via dictionaries by iterating the hashing.
- Choose an iteration method that makes parallelisation of attacks/time-memory trade offs hard to do.
- Many proposals, including scrypt (RFC 7914) and Argon2 (winner of the Password Hashing Competition, see https://password-hashing.net).
- Obtain a trade-off between security and performance of authentication system.
  - Think about Facebook scale operations.
- Typical iteration count: 10,000.

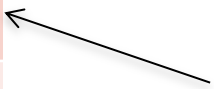
Username	Salt	Password hashes
Alice	B62477A8	AB1134F3DF
Bob	C2EF1377	A76642E3CD
…	…	…

scrypt(`salt` || `password`)

Encryption

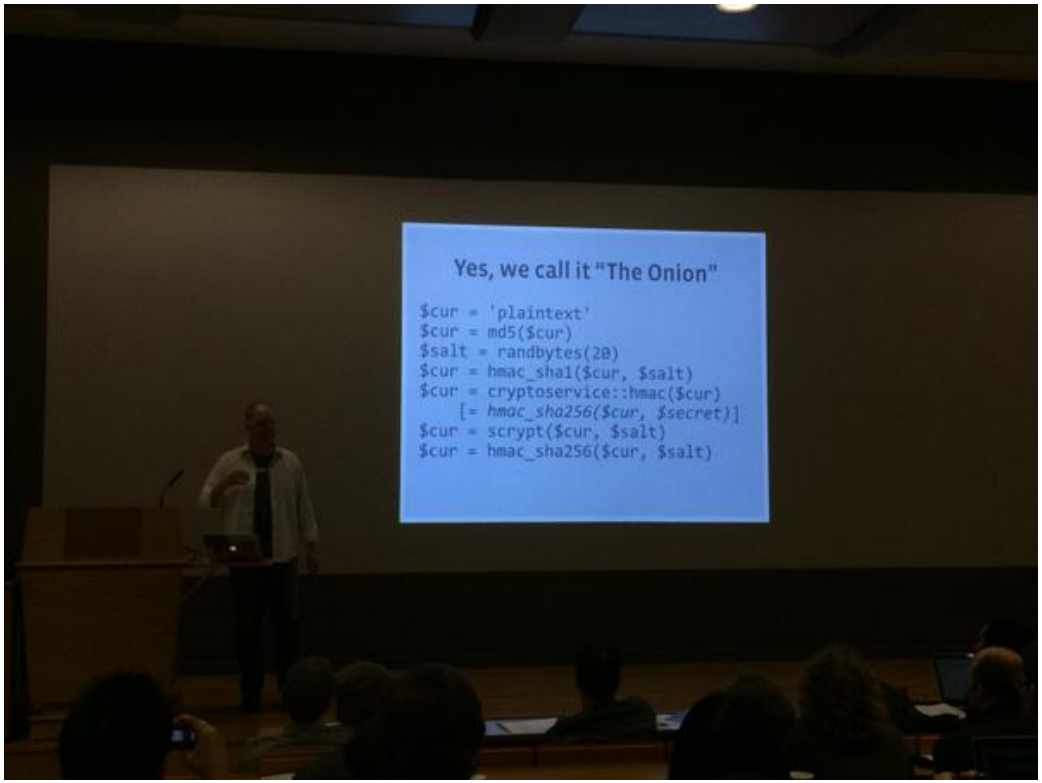
- Why not just encrypt the passwords?
  - Server decrypts entry from database and compares to password sent by user.
- Needs careful selection of encryption method (cf. Adobe's use of ECB mode https://xkcd.com/1286/) and management of key.
- Key needs to be always available and yet heavily protected.
- Can use a special purpose Hardware Security Module (HSM) to store key but HSMs are expensive to buy, complex to manage – more later!

Username	IV	Encrypted passwords
Alice	B62477A8	AB1134F3DF
Bob	C2EF1377	A76642E3CD
…	…	…



$Enc_{K,IV}(\texttt{password})$

Facebook password onion



- Alec Muffet from Facebook explained their approach to password protection at RealWorldCrypto 2015.
- Facebook have multiple layers of protection, mostly for legacy reasons.
- MD5 + HMAC_SHA1 + HMAC (with secret input) + scrypt + HMAC_SHA256!

Further reading

- http://arstechnica.com/security/2013/05/how-crackers-make-minced-meat-out-of-your-passwords/
- http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/
- https://en.wikipedia.org/wiki/PBKDF2
- http://www.tarsnap.com/scrypt.html
- https://password-hashing.net/

Further applications of hash functions

Further applications of hash functions

- Next couple of lectures: we'll see how MAC algorithms (for integrity) use hash algorithms.
- Later lectures: digital signature schemes use hash functions and rely heavily on *collision-resistance* for security (cf. previously mentioned attacks on certificates through hash collisions).
- Even later lectures: key derivation use hash functions to derive one key from another key.
- Commitments:
  - Many cryptographic protocols need participants to commit to values.
  - Can do so by sending hash of chosen value to other participants; relies on one-wayness and collision-resistance.

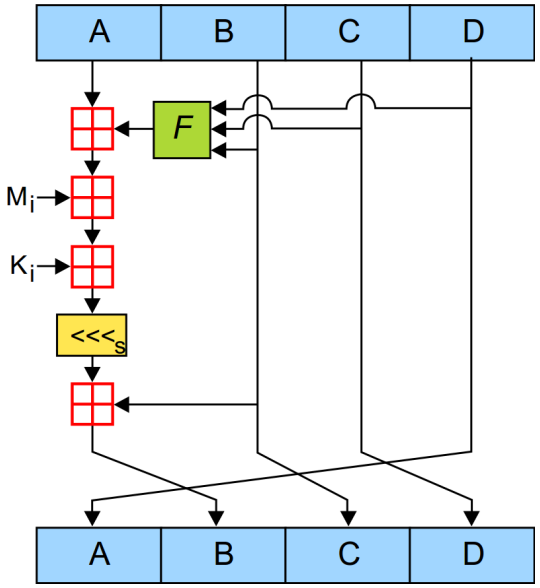
Homework (for week 4)

- **Action 1:** start working on exercise sheet 4.
- **Action 2:** check out earlier labs in preparation for release of first assessed lab (4) on Friday.
- **Action 3:** read Boneh-Shoup – Chapter 8 on hash functions.

Extra slides on MD5

MD5

- Merkle-Damgård hash with output size n = 128.
- Compression function uses (implicitly) a special block cipher in Davis-Meyer mode, k =512.
- MD5 block cipher has a round function operating on 4 x 32-bit words.
- Round function is iterated 64 times; uses mod 2^{32} additions, bit shifts, other simple bit operations (F functions).
- M_i are 16 x 32-bit message subblocks making up 512-bit message block; K_i are round constants (not keys!).
- Each M_i is processed 4 times in 4 different rounds, each time with a different F function.



"MD5" by Surachit - self-made SVG, based on [1] by User:Matt Crypto. Licensed under CC BY-SA 3.0 via Commons - https://commons.wikimedia.org/wiki/File:MD5.svg#/media/File:MD5.svg

Insecurity of MD5

1991: MD5 was introduced to replace weak MD4.

1993,1996: researchers identify initial flaws in MD5, recommend replacement.

2004: researchers show that MD5 is not collision-resistant; proof by example: everybody can reproduce the collisions!

2005: researchers generate X.509 certificates with different keys but matching MD5 hash; very practical and serious security threat.

2008: researchers obtain fake MD5-based certificate from Verisign.

2012: flame malware uses fake certificate issued by Microsoft, exploiting MD5 collisions.

Today: it's possible to produce MD5 collisions in seconds on a laptop.