School of Computing
FACULTY OF ENGINEERING & PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

COMP3911 Secure Computing

3: Message Authentication

Nick Efford

https://comp3911.info

Slides are customised for XJCO3911 by Turgay Celik (Turgay.Celik@swjtu.edu.cn)

Objectives

UNIVERSITY OF LEEDS

- To explore how hash functions work
- To understand how hash functions are used in HMACs, a tool for checking data integrity & authenticity
- To recognise the threats posed by collisions and length extension attacks, and see how these are mitigated

Desirable Goals

UNIVERSITY OF LEEDS

We would like to

- Detect whether a piece of data has been altered
- Be assured that the data is authentic (e.g., was created by the person claiming to have created it)
- Prevent others from making sense of the data, even if they obtain access to it

Practical Example

Subresource integrity checking for websites:

```
<script src="https://code.jquery.com/jquery-2.1.4.min.js"
  integrity="sha256-8WqyJLuWKRBVhxXIL1jBDD7SDxU936oZkCnxQbWwJVw=">
</script>
```

what’s this?

Hash Functions

UNIVERSITY OF LEEDS

- Applying *H* produces a fixed-length **message digest** or **hash** from any length of input, *x*
- For any *x*, *H*(*x*) is relatively easy to compute
- **Avalanche effect**: changing just a single bit anywhere in *x* produces a large change in *H(x)*

```
MD5("aaaa") = "74b87337454200d4d33f80c4663dc5e5"
MD5("aaab") = "4c189b020ceb022e0ecc42482802e2b8"
```

Required Properties

UNIVERSITY OF LEEDS

- **Pre-image resistance**: given hash h, it is computationally infeasible to find x such that $H(x) = h$
- **Second pre-image resistance**: given input x, it is computationally infeasible to find input y such that $y \neq x$ and $H(y) = H(x)$
- **Collision resistance**: it is computationally infeasible to find any pair of different inputs $\{x, y\}$ for which $H(x) = H(y)$

The Birthday Bound

UNIVERSITY OF LEEDS

How many randomly-chosen people need to be in a room together before there is a ~50% chance that two of them will share the same birthday?

Answer: **23** 😲

In general, if we are generating strings randomly from a space of 2^n possibilities, we can expect a 50% chance of finding a collision after having generated $2^{n/2}$ strings…

… so if there are 2^{128} possible hashes, 2^{64} operations will be enough to have a good chance of finding a collision!

Standard Hash Functions

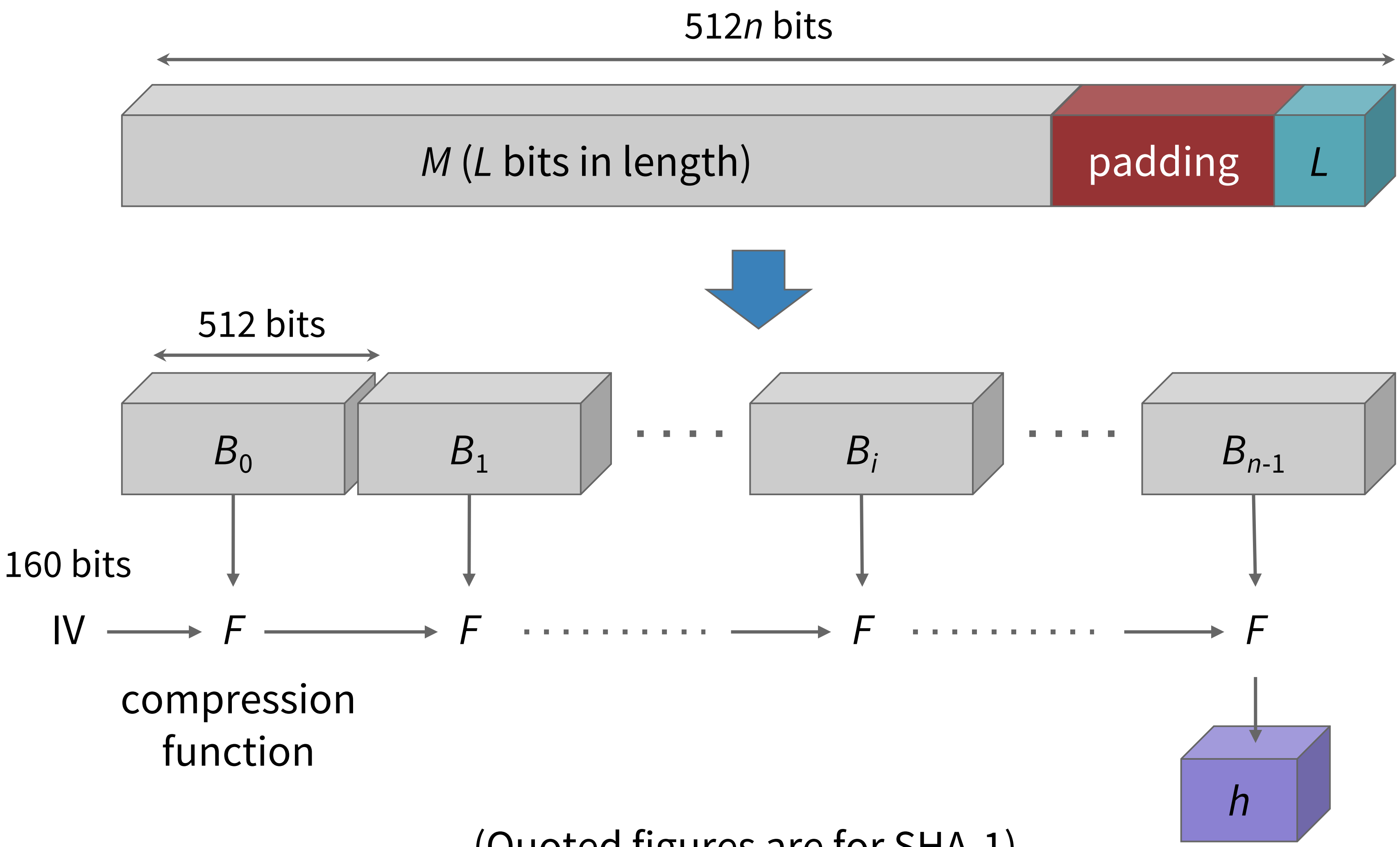
UNIVERSITY OF LEEDS

	Function	Output (bits)	Block Size (bits)	No. of Rounds	Security (bits)
DO NOT USE!	MD5	128	512	64	≤18*
	SHA-1	160	512	80	<63*
SHA-2 family	SHA-224	224	512	64	112
	SHA-256	256	512	64	128
	SHA-384	384	1024	80	192
	SHA-512	512	1024	80	256

These functions all use the **Merkel-Damgård construction**

Merkle-Damgård Construction

UNIVERSITY OF LEEDS



(Quoted figures are for SHA-1)

Compression Function

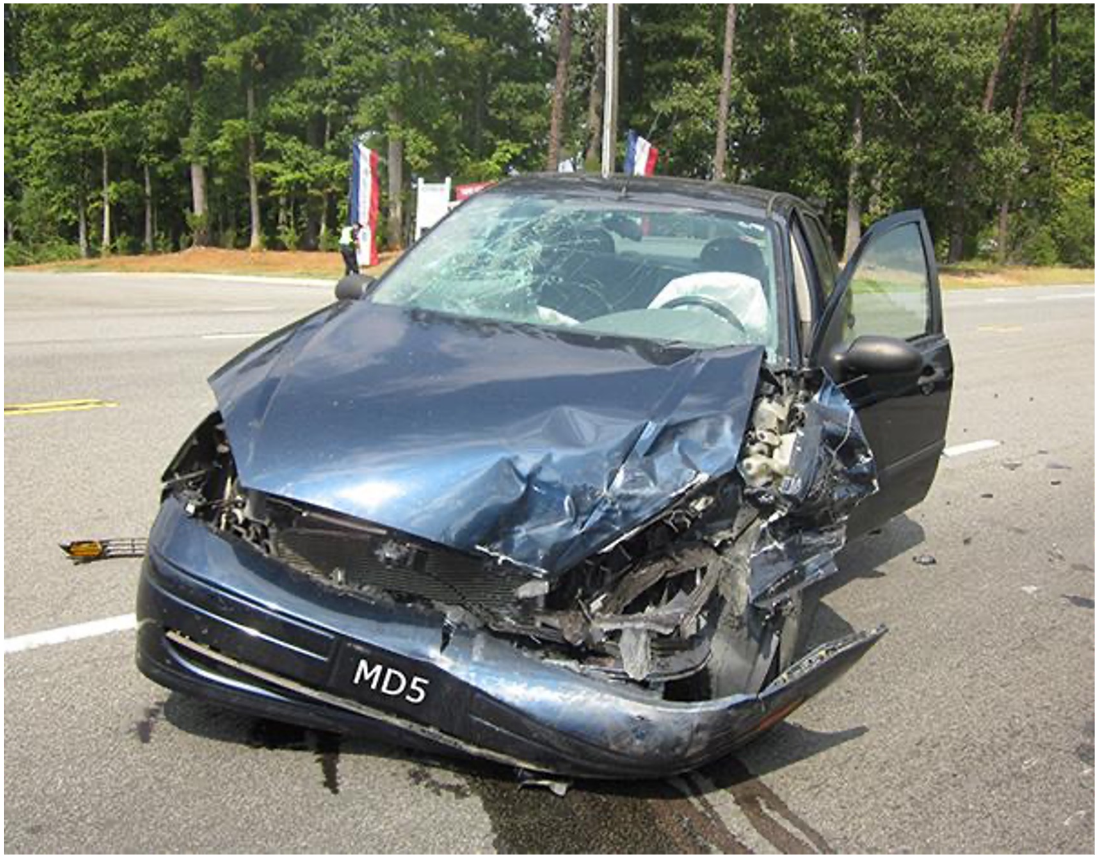
UNIVERSITY OF LEEDS

- Compression function is a specialized **block cipher**
- Message blocks are used as the cipher key
- Cipher encrypts the previous CF output value, and result of encryption is XORed with that previous value to yield the new output value

$$F_i = E_i(F_{i-1}) \oplus F_{i-1}$$

- No such previous value exists for first message block – so we use a fixed **Intialization Vector** (IV) here

MD5 Collisions



What About SHA-1?

UNIVERSITY OF LEEDS

- Deprecated by NIST in 2011, and no longer accepted in TLS certificates by web browsers
- The SHAppening (2015): estimated cost of finding a SHA-1 collision with Amazon EC2 as $75K – $120K
- Shattered.io (2017): found the first realistic collision for actual documents (PDFs)
- Leurent & Peyrin (2019): possible to find ‘chosen prefix’ SHA-1 collisions for ~$100K
    - https://eprint.iacr.org/2019/459
    - https://github.com/Cryptosaurus/sha1-cp

spookeevee
@eevee

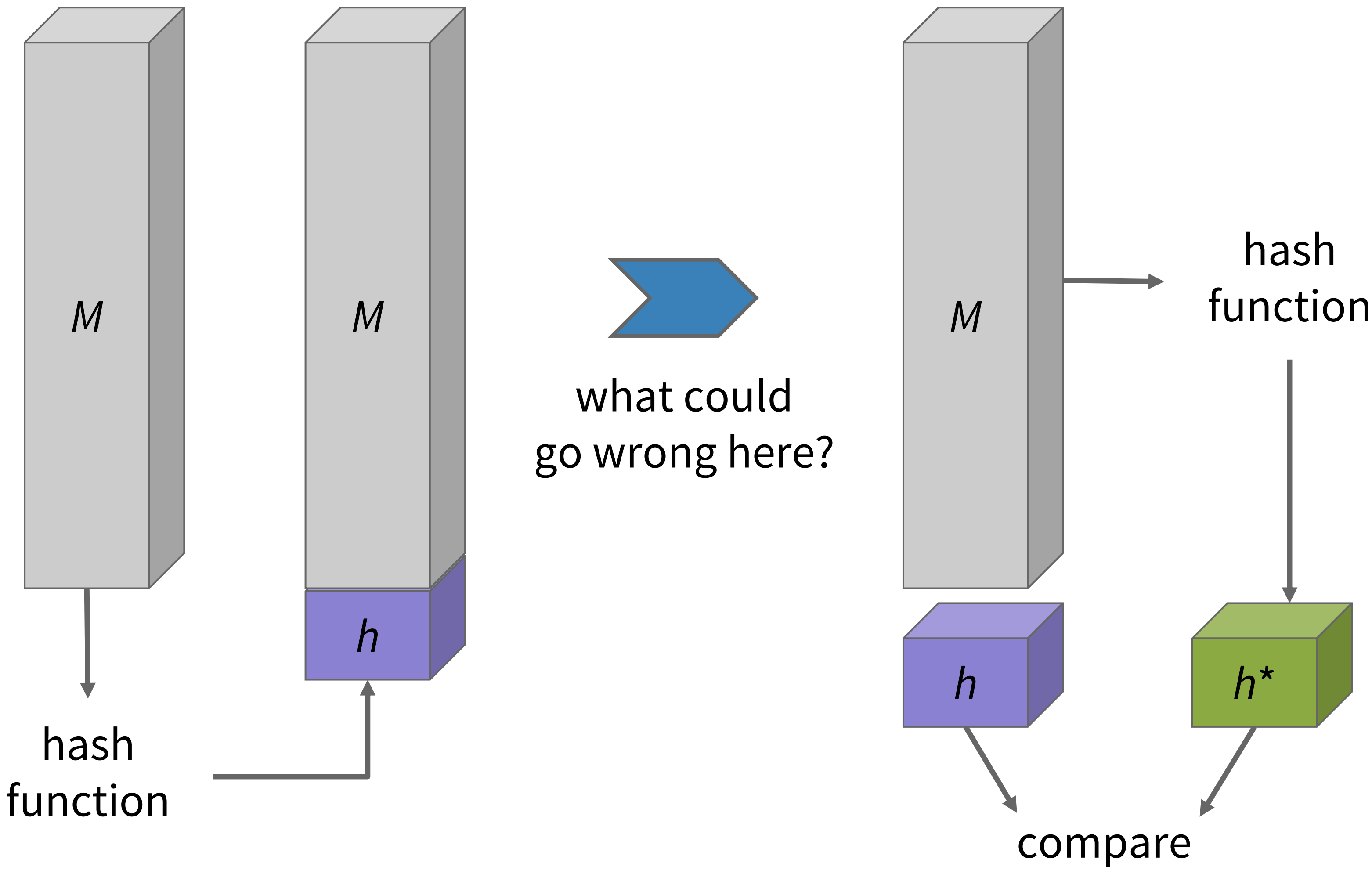
THEY USED A HASH
They used a SHA1 hash
A SHA1 HASH
Implemented in Flash
A SHA1 HASH
It was easy to smash
A SHA1 HASH
Now I got all your cash

6:15pm · 2 Oct 2016 · Twitter for Android

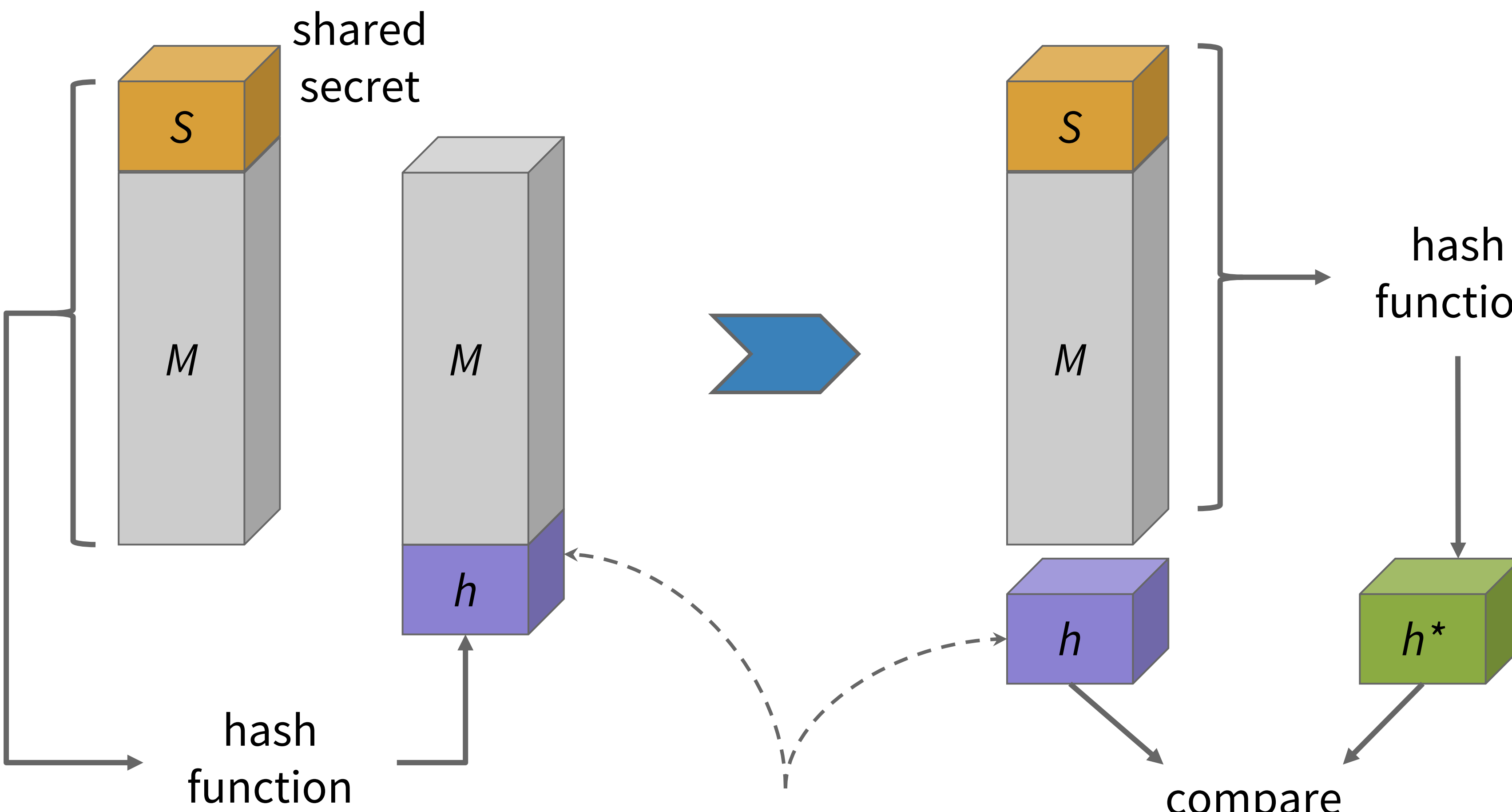
657 RETWEETS **1,188** LIKES

Using Hash Functions

UNIVERSITY OF LEEDS



A Solution: MACs

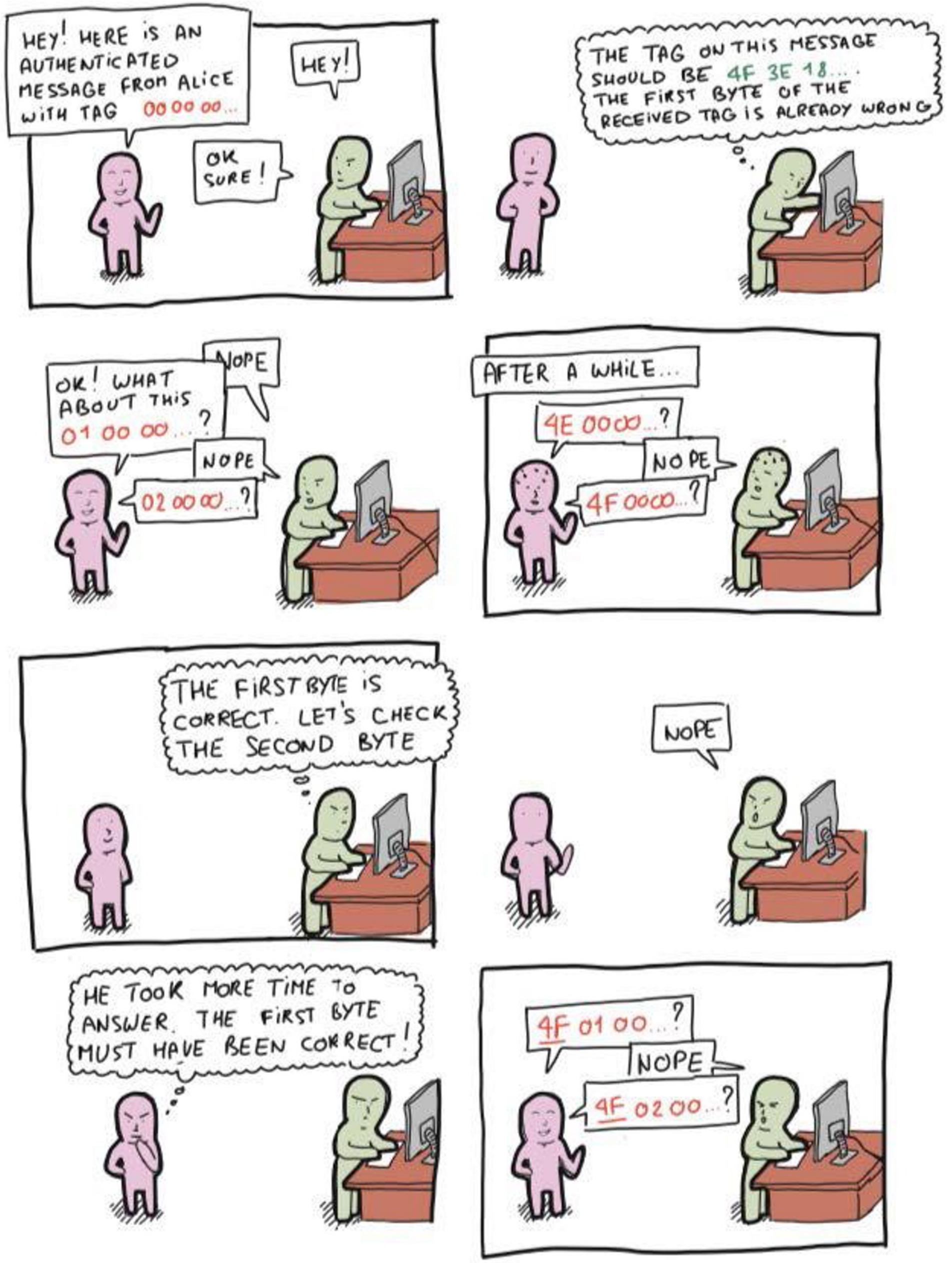


we call this a **message authentication code** (MAC) or **authentication tag**

Issues

UNIVERSITY OF LEEDS

- How do we choose a good-quality secret?
- How do we share the secret securely?
- How do we compare authentication tags securely?
- How do we prevent **replay attacks**?
- Many of the Merkle-Damgård hash functions are vulnerable to **length extension attacks**

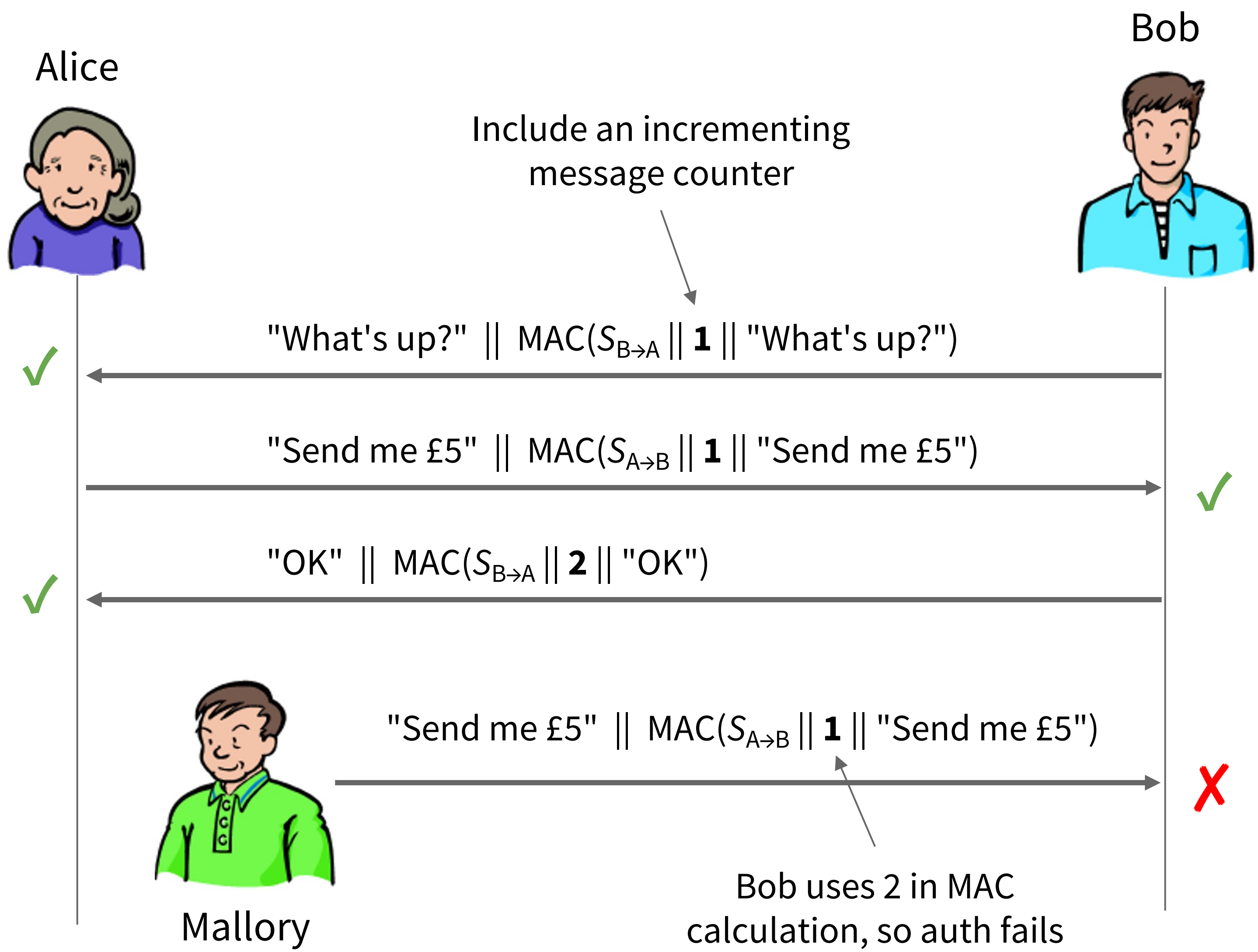


[Figure from Wong, *Real-World Cryptography*]

Auth tag comparison needs to use a **constant-time algorithm**

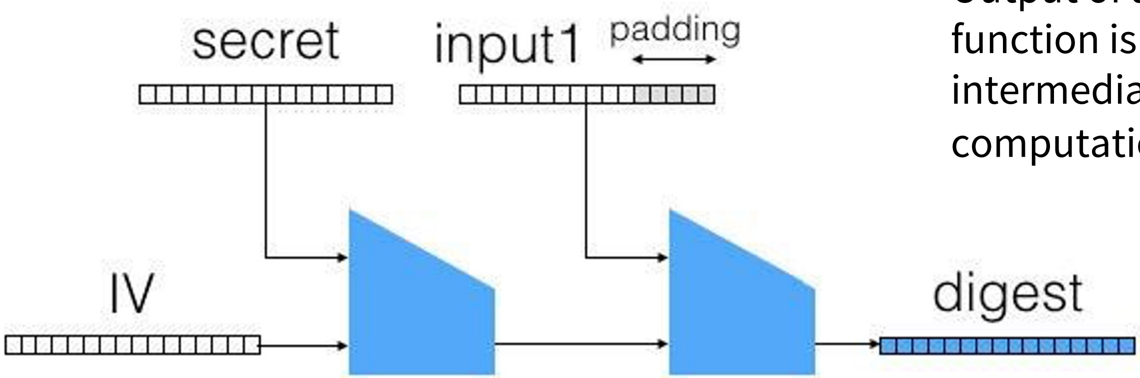
If the comparison 'returns early', an attacker can measure response times and reconstruct a valid tag byte-by-byte…

Preventing Replay Attacks

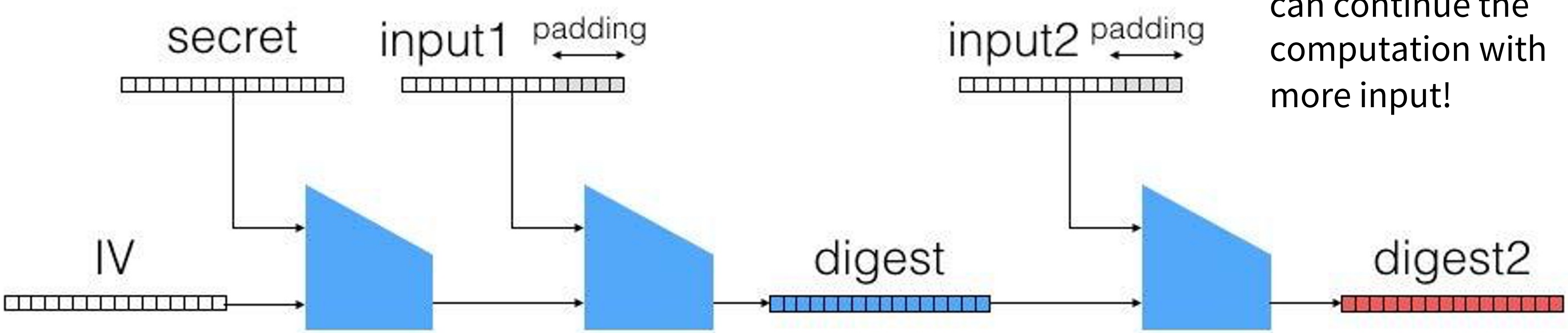


Length Extension Attack

UNIVERSITY OF LEEDS



Output of a Merkle-Damgård hash function is equivalent to the full intermediate state of hash function computation at that point...



… which means we can continue the computation with more input!

[Figures from Wong, *Real-World Cryptography*]

Length Extension Attack

UNIVERSITY OF LEEDS

- Auth tag for message M_1 given by $H(S \,||\, M_1 \,||\, P_1)$
- **But we can extend hash computation**, feeding it M_2, to which padding P_2 is added…
- … which yields $H(S \,||\, M_1 \,||\, P_1 \,||\, M_2 \,||\, P_2)$
- … **which has the same value** as a tag computed for the message $M_1 \,||\, P_1 \,||\, M_2$
- Attack succeeds if you can engineer things so $M_1 \,||\, P_1 \,||\, M_2$ is interpreted as a valid message
- Example: 2009 Flickr API vulnerability

Solution: HMAC

UNIVERSITY OF LEEDS

- Uses a **nested hashing** approach
- Derive **inner secret** S_{IN} & **outer secret** S_{OUT} by padding or hashing secret S to the size of a block, then XORing with constants
- Then compute

$$\mathrm{HMAC}(S, M) = \underbrace{H(S_{OUT} \,||\, \overbrace{H(S_{IN} \,||\, M)}^{\text{inner hash}})}_{\text{outer hash}}$$

SHA-3

UNIVERSITY OF LEEDS

- Result of a NIST-sponsored competition to find a new standard based on entirely different principles to Merkle-Damgård functions
- Winning entry, Keccak, was announced in 2012 and became a formal new standard in 2015
- Offers same sized outputs as SHA-2, so can act as a drop-in replacement if SHA-2 suddenly becomes vulnerable
- Part of the internal state never leaks into the computed hash, so SHA-3 is **not vulnerable to LE attacks**

Summary

UNIVERSITY OF LEEDS

We have

- Explored the properties required of hash functions
- Examined a range of standard hash functions, including the obsolete functions MD5 & SHA-1
- Discussed the risks posed by hash function collisions and length extension attacks
- Seen how a shared secret is used with a hash function in the HMAC algorithm, and how this helps us to check the integrity & authenticity of data

Follow-Up / Further Reading

UNIVERSITY OF LEEDS

- Code Examples
- Exercises 1–4
- MD5 considered harmful: creating a rogue CA certificate
- MD5 length extension attack on Flickr API
    - Advisory and example code
- Poisonous MD5 – Wolves Among The Sheep