Crypto III

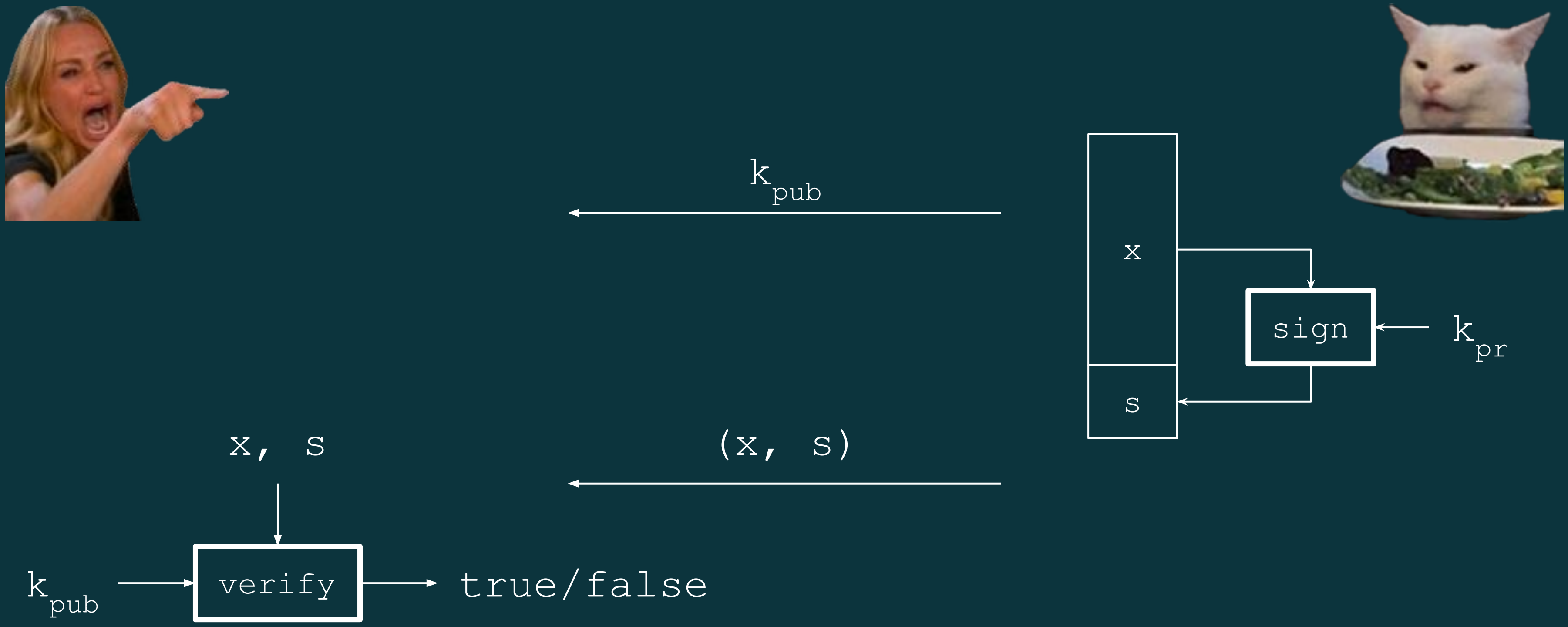
Kuruwa

ToC

Motivation

- Bob orders an RTX-3090 from Alice
- After seeing the RTX-3090, Bob states that he has never ordered it
- How can Alice prove towards a judge that Bob has ordered an RTX-3090? (And that she did not fabricate the order herself)
  - Symmetric cryptography fails because both Alice and Bob can be malicious
  - Can be achieved with public-key cryptography

Digital Siganture



Main Idea

- For a given message x, a digital signature is appended to the message (just like a conventional signature)
- Only the person with the private key should be able to generate the signature
- The signature must change for every document
    - The signature is realized as a function with the message x and the private key as input
    - The public key and the message x are the inputs to the verification function

Objectives

- Integrity
    - Ensures that a message has not been modified in transit.
- Message Authentication
    - Ensures that the sender of a message is authentic. An alternative term is data origin authentication.
- Non-repudiation
    - Ensures that the sender of a message can not deny the creation of the message. (e.x. order of a GPU)

RSA Signature

- To generate the signature
  - Sign (encrypt) the message x with the private key

$$s = sig_{Kpr}(x) = x^d \bmod n$$

  - Append s to message x
- To verify the signature
  - Verify (decrypt) the signature with the public key

$$x' = ver_{Kpub}(s) = s^e \bmod n$$

  - If x = x', the signature is valid

RSA Signature Protocol

$k_{pub} = (n, e)$
$k_{pr} = d$

$\xleftarrow{\quad k_{pub} \quad}$

$s = x^d \bmod n$

$\xleftarrow{\quad (x, s) \quad}$

$x' = s^e \bmod n$
if $x' = x \longrightarrow$ valid
if $x' \neq x \longrightarrow$ invalid

Existential Forgery

$k_{pub} = (n, e)$
$k_{pr} = d$

(n, e) ←

Choose signature
$s \in \mathbb{Z}_n$
Compute message
$x = s^e \bmod n$

← (n, e)

← (x, s)

Verification:
$x' = s^e = x \bmod n$
$\longrightarrow$ Signature is valid

Existential Forgery

- An attacker can generate valid message-signature pairs (x, s)
- But an attack can only choose the signature s and NOT the message x
- Formatting the message x according to a **padding scheme** can be used to make sure that an attacker cannot generate valid (x, s) pairs

Digital Signature Algorithm (DSA)

- Key generation of DSA:
  - Generate a prime p with $2^{1023} < p < 2^{1024}$
  - Find a prime divisor q of $p - 1$ with $2^{159} < q < 2^{160}$
  - Find an integer α with $ord(\alpha) = q$
    - $\alpha = g^{(p-1)/q} \neq 1 \bmod p$
  - Choose a random integer d with $0 < d < q$
  - Compute $\beta = \alpha^{d} \bmod p$
- The keys are: $k_{pub} = (p, q, \alpha, \beta)$ and $k_{pr} = (d)$

Digital Signature Algorithm (DSA)

- Signature (message: $H < q$)
  - Choose an integer k_E as a random ephemeral key with $0 < k_E < q$
  - Compute $r = (\alpha^{k_E} \bmod p) \bmod q$
  - Compute $s = k_E^{-1}(H + d \times r) \bmod q$
    - In practice, H is hash of the message
- Verification
  - Compute auxiliary value $u_1 = s^{-1} \times H \bmod q$
  - Compute auxiliary value $u_2 = s^{-1} \times r \bmod q$
  - Compute $v = (\alpha^{u_1} \times \beta^{u_2} \bmod p) \bmod q$
    - if $v = r \longrightarrow$ siganature is valid
    - if $v \neq r \longrightarrow$ siganature is invalid

Correctness

$$s = (H + d \times r) k_E^{-1} \mod q$$

$$\Leftrightarrow k_E = s^{-1} \times H + d(s^{-1} \times r) \mod q$$

$$\Leftrightarrow k_E = u_1 + du_2 \mod q$$

$$\Leftrightarrow \alpha^{k_E} \mod p = \alpha^{u_1 + du_2} \mod p$$

$$\Leftrightarrow (\alpha^{k_E} \mod p) \mod q = (\alpha^{u_1} \times \beta^{u_2} \mod p) \mod q$$

$$\Leftrightarrow r = v$$

Security

- DSA can achieve same security level as RSA scheme with less siganture length

p	q	length	security
1024	160	320	80
2048	224	448	112
3072	256	512	128

ECDSA

- Key generation of ECDSA:
  - Find a generator G on an elliptic curve E with prime order n
  - Choose a random integer d with $0 < d < n$
  - Compute $P = dG$
- The keys are: $k_{pub} = (E, G, n, P)$ and $k_{pr} = (d)$
  - Shorter private key and higher speed than DSA

ECDSA

- Signature (message: $H < n$)
  - Choose an integer k_E as a random ephemeral key with $0 < k_E < n$
  - Calculate the curve point $(x_1, y_1) = k_E \times G$
  - Compute $r = x_1 \bmod n$
  - Compute $s = k_E^{-1}(H + d \times r) \bmod n$
- Verification
  - Compute auxiliary value $u_1 = s^{-1} \times H \bmod n$
  - Compute auxiliary value $u_2 = s^{-1} \times r \bmod n$
  - Compute $(x_1, y_1) = u_1G + u_2P$
    - if $x_1 = r \bmod n \longrightarrow$ siganature is valid
    - if $x_1 \neq r \bmod n \longrightarrow$ siganature is invalid

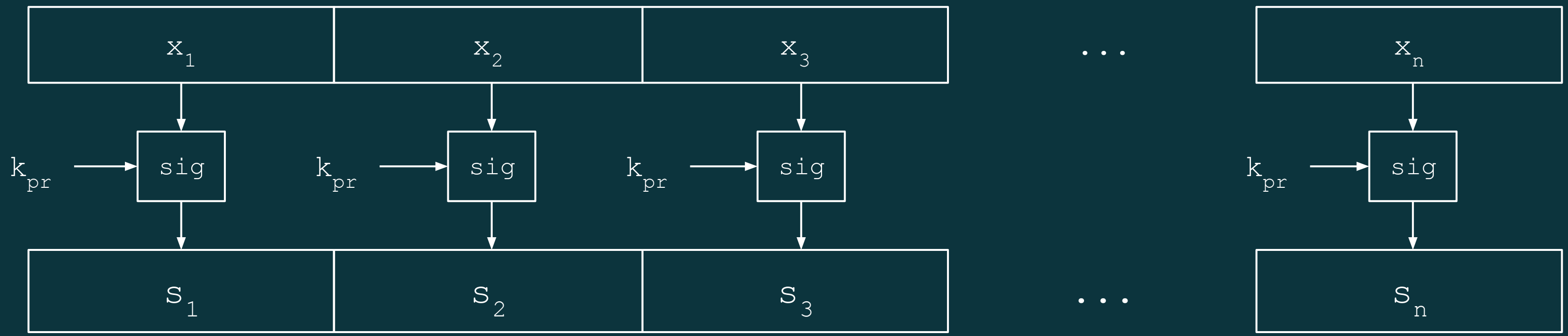
Sensitivity

- The entropy of the random value k_E are critical
- Example: sign two different messages, $k_1 = k_2$
  - $k_1 = s_1^{-1}H_1 + d(s_1^{-1}r_1) \bmod q$
  - $k_2 = s_2^{-1}H_2 + d(s_2^{-1}r_2) \bmod q$
  - $d = (s_1^{-1}H_1 - s_2^{-1}H_2) / (s_2^{-1}r_2 - s_1^{-1}r_1)$

Hash

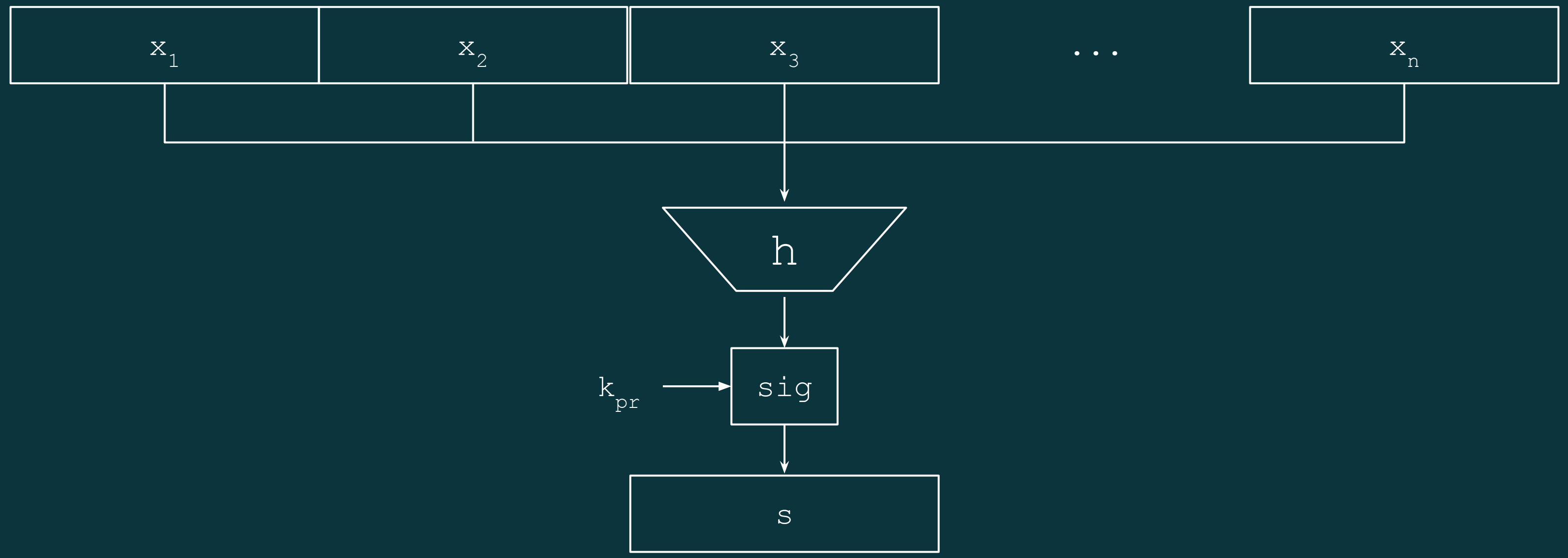
Hash - Motivation

- Naive signing of long messages generates a signature of same length.

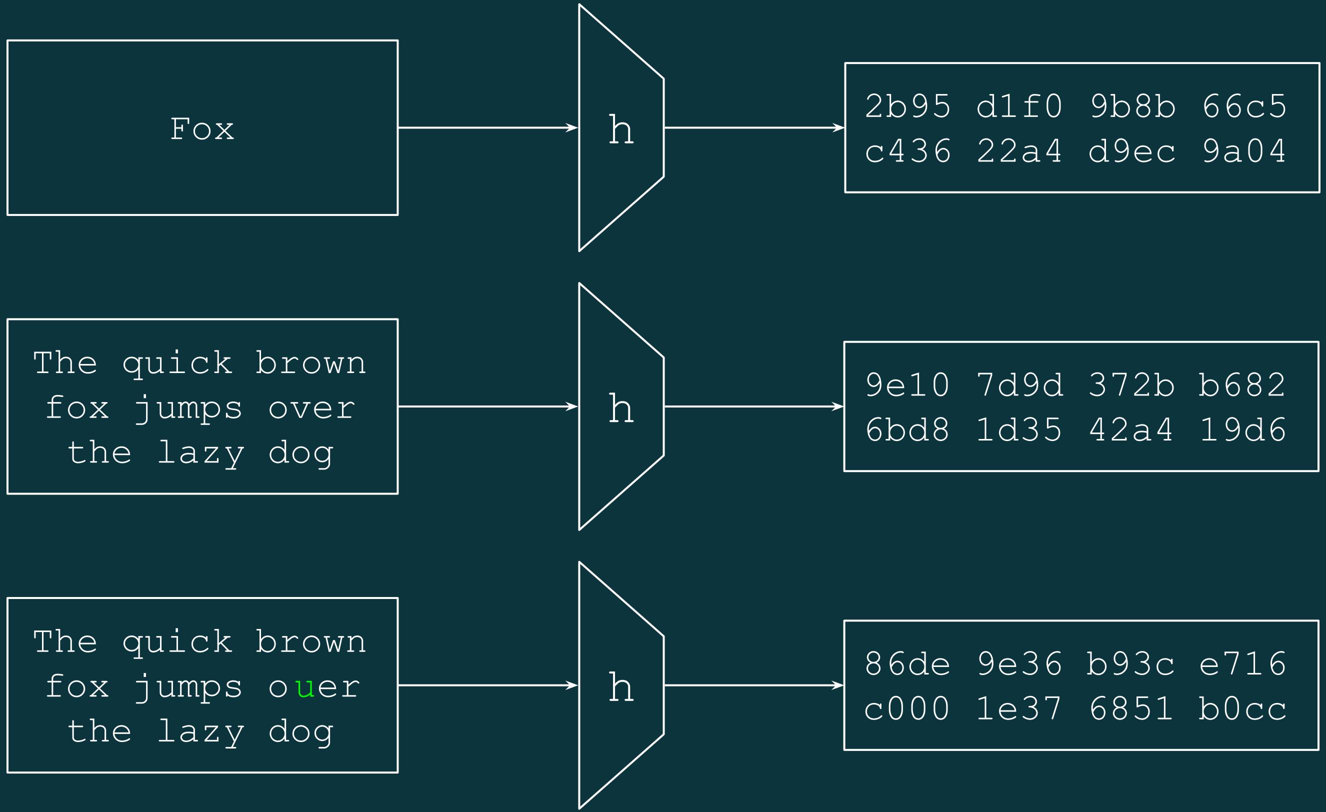


- Solution
  - Instead of signing the whole message, sign only a digest (hash)

Digital Signature with Hash Function

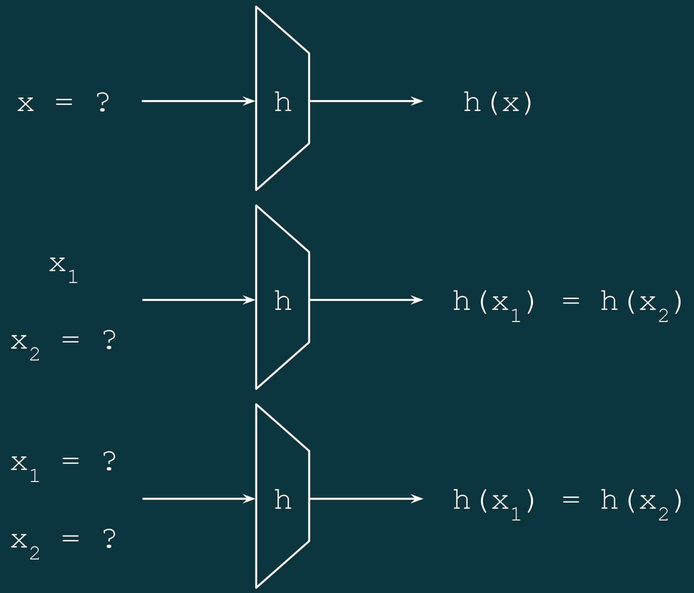


Avalanche Effect



Security Properties

- Pre-image resistance
  - For a given output z, it is computationally infeasible to find any input x such that $h(x) = z$
- Second pre-image resistance
  - Given x_1, and thus $h(x_1)$, it is computationally infeasible to find any x_2 such that $h(x_1) = h(x_2)$
- Collision resistance
  - It is computationally infeasible to find any pairs $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$

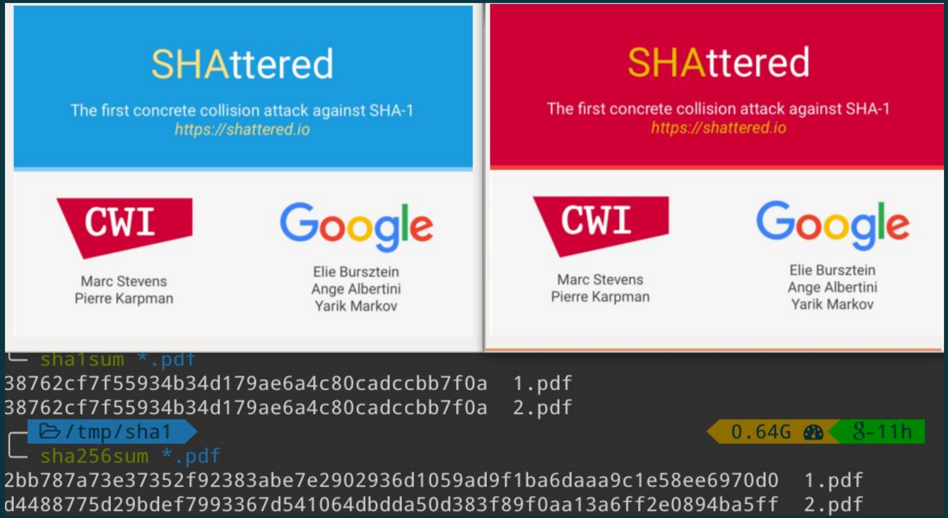


Birthday Paradox

- How hard is it to find a collision with a probability of 0.5?
- Related problem: How many people are needed such that two of them have the same birthday with a probability of 0.5?
- Far fewer than 365/2 = 182.5! This is called the birthday paradox (Search takes ≈ $\sqrt{n}$ steps)
- To deal with this paradox, hash functions need a output size of at least 160 bits

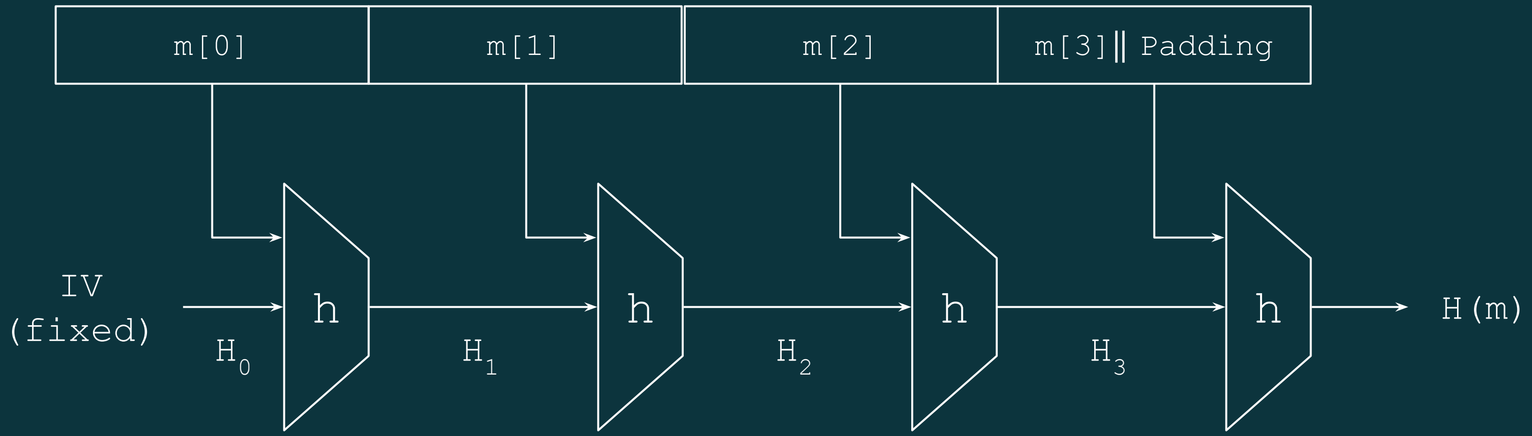
SHA-1 Collision

- In 2017 Google presented 2 PDF files that display different content, yet have the same SHA-1 digest.
  - Took about 2^{63} SHA1 computations



Merkle-Damgård construction

- Used in the design of many popular hash algorithms such as MD5, SHA-1 and SHA-2



Length Extension Attack

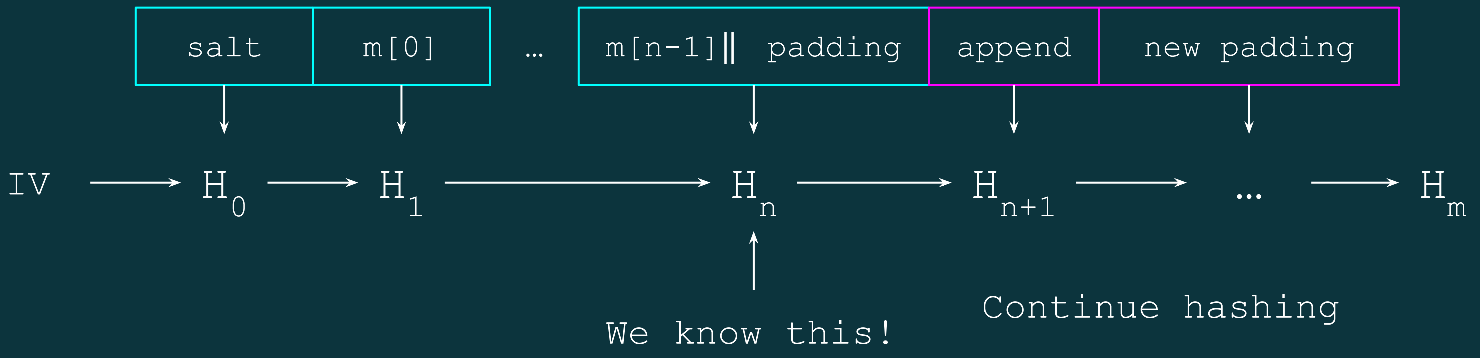
- When a Merkle-Damgård based hash is misused as a message authentication code with construction

$$H(salt \,\|\, message)$$

- If message and the length of salt is known, we can include extra information and forge a valid hash

Length Extension Attack

- Continue calculating hash after appending extra message
- New plaintext is message|| padding|| append



HashPump

- CRC32, MD5, SHA1, SHA256 and SHA512 support

```
└─# hashpump -s '6d5f807e23db210bc254a28be2d6759a0f5f5d99' --data 'count=10&lat=37.351&user_id=1&long=-11
9.827&waffle=eggo' -a '&waffle=liege' -k 14
```

new hash

```
0e41270260895979317fff3898ab85668953aaa2
```

new messsage

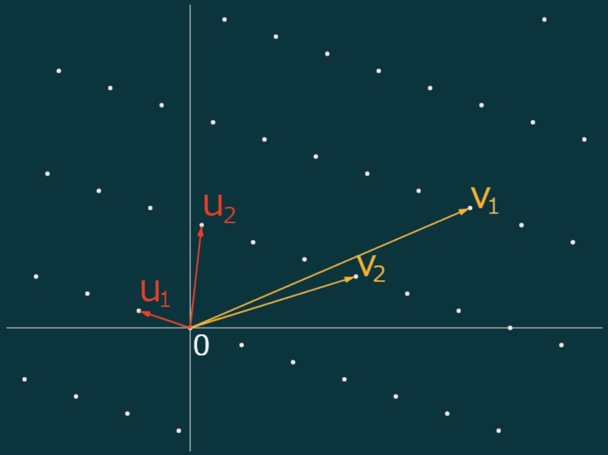
```
count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02(&waffle=liege
```

Lattices

Lattices

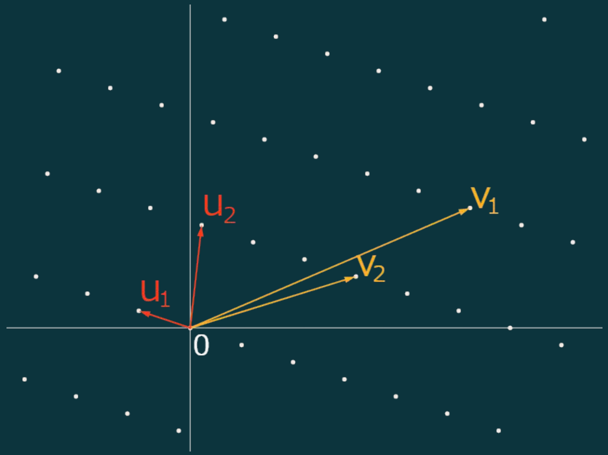
- Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \in \mathbb{R}^m$ be a set of linearly independent vectors
- The **lattice** L generated by $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ is the set of linear combinations with coefficients in $\mathbb{Z}$,

$$L = \{a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \ldots + a_n\mathbf{v}_n \mid a_1, a_2, \ldots, a_n \in \mathbb{Z}\}$$



Shortest Vector Problem (SVP)

- The basis of lattice is not unique
- Given a basis of L, find the shortest vector in L
  - SVP is NP-hard



A Congruential PKC

- A toy model of a real public key cryptosystem is described
- It turns out to have an unexpected connection with lattices of dimension 2
- An example of how lattices may appear in cryptanalysis even when the underlying hard problem appears to have nothing to do with lattices

Key Creation

- Alice chooses a large positive integer q as public parameter, and two other secret positive integers f and g, satisfying
  - $f < \sqrt{q/2}$, $\sqrt{q/4} < g < \sqrt{q/2}$, and gcd(f , qg) = 1
- Then Alice compute $h \equiv f^{-1}g \pmod{q}$, with $0 < h < q$
- Public key: (q, h)
- Secret key: (f, g)

Encryption

- To send a message m, Bob chooses a random integer r, with
  - $0 < m < \sqrt{q/4}$
  - $0 < r < \sqrt{q/2}$
- The ciphertext is $e \equiv rh + m \pmod{q}$, with $0 < e < q$

Decryption

- Alice decrypts ciphertext e by computing
  - $a \equiv fe \pmod{q}$
  - $b \equiv f^{-1}a \pmod{g}$
- Then b is the plaintext m

Correctness

- $a \equiv fe \equiv f(rh + m) \equiv frf^{-1}g + fm \equiv rg + fm \pmod{q}$
- The size restrictions on f, g, r, m imply that

$$rg + fm < \sqrt{\frac{q}{2}}\sqrt{\frac{q}{2}} + \sqrt{\frac{q}{2}}\sqrt{\frac{q}{4}} < q$$

- So Alice can get the exact value $a = rg + fm$
- Then Alice computes

$$b \equiv f^{-1}a \equiv f^{-1}(rg + fm) \equiv f^{-1}fm \equiv m \pmod{g}$$

- Since $m < \sqrt{q/4} < g$, it follows that $b = m$

Overall Process

Choose a modulus q

Choose f, g with restrictions

Compute h $\equiv f^{-1}g \pmod{q}$

$\xleftarrow{(q,\ h)}$

Choose m $< \sqrt{q/4}$

Choose r $< \sqrt{q/2}$

Compute e $\equiv rh + m \pmod{q}$

$\xrightarrow{e}$

Compute a $\equiv fe \pmod{q}$

Compute b $\equiv f^{-1}a \pmod{g}$

Then b is the plaintext m

Example

- Alice chooses
  - $q = 122430513841$
  - $f = 231231 \approx 0.66\sqrt{q}$
  - $g = 195698 \approx 0.56\sqrt{q}$
- Alice computes
  - $f^{-1} \equiv 49194372303 \pmod{q}$
  - $h \equiv f^{-1}g \equiv 39245579300 \pmod{q}$
- Public key: $(q, h) = (122430513841, 39245579300)$
- Bob chooses
  - message $m = 123456$
  - random value $r = 101010$
- Bob computes ciphertext $e \equiv rh + m \equiv 18357558717 \pmod{q}$
- To decrypt, Alice computes
  - $a \equiv fe \equiv 48314309316 \pmod{q}$
  - $b \equiv f^{-1}a \equiv 193495 \times 48314309316 \equiv 123456 \equiv m \pmod{g}$

Cryptanalysis

- Brute-force search: O(q) operations
- If attacker can find any pair of positive integers F and G s.t.
  - $Fh \equiv G \pmod q$
  - $F, G = O(\sqrt{q})$

  then (F, G) is likely to serve as a decryption key
- Rewriting $Fh = G + qR$, we reformulate Eve's task as that of finding a pair of comparatively small integers (F, G) with

unknown integers

$$F(1,h) - R(0,q) = (F,G)$$

known vectors

unknown small vector

Cryptanalysis (cont.)

- Thus attacker knows two vectors $\mathbf{v}_1 = (1, h)$ and $\mathbf{v}_2 = (0, q)$, both of length $O(q)$
- Attacker wants to find a linear combination $w = a_1\mathbf{v}_1 + a_2\mathbf{v}_2$ such that w has length $O(\sqrt{q})$
- This corresponds to find a short nonzero vector in the set

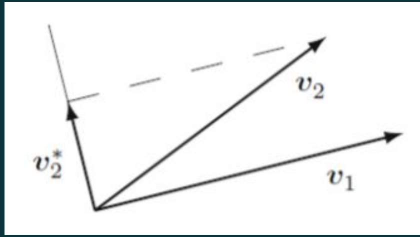
$$L = \{a_1\mathbf{v}_1 + a_2\mathbf{v}_2 : a_1, a_2 \in \mathbb{Z}\}$$

- This set L is an example of a two-dimensional lattice
- Unfortunately for Bob and Alice, there is an extremely rapid method for finding short vectors in two-dimensional lattices

Gaussian Lattice Reduction

- Suppose that $L \subset \mathbb{R}^2$ is a 2-dimensional lattice with basis vectors $\mathbf{v}_1$, $\mathbf{v}_2$
  - May assume $\|\mathbf{v}_1\| < \|\mathbf{v}_2\|$
- If allowed to subtract any multiple of $\mathbf{v}_1$, then replace $\mathbf{v}_2$ with the vector

$$v_2^* = v_2 - \frac{v_1 \cdot v_2}{\|v_1\|^2} v_1$$



  - $\mathbf{v}_2^*$ is orthogonal to $\mathbf{v}_1$
  - But $\mathbf{v}_2^*$ is unlikely to be in L
- So the best is to replace $\mathbf{v}_2$ with the vector $\mathbf{v}_2 - m\mathbf{v}_1$ with

$$m = \left\lfloor \frac{v_1 \cdot v_2}{\|v_1\|^2} \right\rceil$$

Gaussian Lattice Reduction (cont.)

- If $\|\mathbf{v}_1\| < \|\mathbf{v}_2\|$, then stop
- Otherwise, swap $\mathbf{v}_1$ and $\mathbf{v}_2$ and repeat the process

Loop

  If $\|v_2\| < \|v_1\|$, swap v_1 and v_2.

  Compute $m = \lceil v_1 \cdot v_2 / \|v_1\|^2 \rfloor$.

  If $m = 0$, return the basis vectors v_1 and v_2.

  Replace v_2 with $v_2 - mv_1$.

Continue Loop

- When the algorithm terminates
  - The vector $\mathbf{v}_1$ is a shortest nonzero vector in L
  - The algorithm solves SVP

Lenstra-Lenstra-Lovász Algorithm (LLL)

- Given a lattice L, LLL solves approximated SVP in polynomial time
- The shortest vector $\mathbf{v}$ it found satisfies
  $$\|\mathbf{v}\| \leq 2^{(n-1)/4}|\det L|^{1/n}$$
- On average, LLL achieves
  $$\|\mathbf{v}\| \leq 1.02^{n}|\det L|^{1/n}$$

INPUT

a lattice basis $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{Z}^m$

a parameter δ with $\frac{1}{4} < \delta < 1$, most commonly $\delta = \frac{3}{4}$

PROCEDURE

$\mathbf{B}^* \leftarrow \mathrm{GramSchmidt}(\{\mathbf{b}_0, \ldots, \mathbf{b}_n\}) = \{\mathbf{b}_0^*, \ldots, \mathbf{b}_n^*\}$; *and do not normalize*

$\mu_{i,j} \leftarrow \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}$; *using the most current values of $\mathbf{b}_i$ and $\mathbf{b}_j^*$*

$k \leftarrow 1$;

while $k \leq n$ **do**

    for j **from** $k-1$ **to** 0 **do**

        if $|\mu_{k,j}| > \frac{1}{2}$ **then**

            $\mathbf{b}_k \leftarrow \mathbf{b}_k - \lfloor \mu_{k,j} \rceil \mathbf{b}_j$;

            Update $\mathbf{B}^*$ and the related $\mu_{i,j}$'s as needed.

            (The naive method is to recompute $\mathbf{B}^*$ whenever $\mathbf{b}_i$ changes:

            $\mathbf{B}^* \leftarrow \mathrm{GramSchmidt}(\{\mathbf{b}_0, \ldots, \mathbf{b}_n\}) = \{\mathbf{b}_0^*, \ldots, \mathbf{b}_n^*\}$;)

        end if

    end for

    if $\langle \mathbf{b}_k^*, \mathbf{b}_k^* \rangle \geq \left(\delta - \mu_{k,k-1}^2\right) \langle \mathbf{b}_{k-1}^*, \mathbf{b}_{k-1}^* \rangle$ **then**

        $k \leftarrow k+1$;

    else

        Swap $\mathbf{b}_k$ and $\mathbf{b}_{k-1}$;

        Update $\mathbf{B}^*$ and the related $\mu_{i,j}$'s as needed.

        $k \leftarrow \max(k-1, 1)$;

    end if

end while

return $\mathbf{B}$ the LLL reduced basis of $\{b_0, \ldots, b_n\}$

OUTPUT

the reduced basis $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{Z}^m$

Coppersmith's Method

- **Input:** $f(x) \in \mathbb{Z}[x]$, $N \in \mathbb{Z}$
- **Output:** r s.t. $f(r) \equiv 0 \bmod N$
- **Intermediate output:** $Q(x)$ such that $Q(r) = 0$ over $\mathbb{Z}$
  - $Q(x) = s(x)f(x) + t(x)N$
  - $Q(r) \equiv 0 \bmod N$ by construction
  - If $|r| \leq R$, then we can bound

$$|Q(r)| = |Q_3 r^3 + Q_2 r^2 + Q_1 r + Q_0|$$

$$\leq |Q_3|R^3 + |Q_2|R^2 + |Q_1|R + |Q_0|$$

  - If $|Q(r)| < N$ and $Q(r) \equiv 0 \bmod N$, then $Q(r) = 0$
- We want a Q in our lattice with short coefficient vector!

Coppersmith's Method

1. Construct a matrix of coefficient vectors of elements of <f(x), N>
2. Run LLL algorithm on this matrix
3. Construct a polynomial Q from the shortest vector output
4. Factor Q to find its roots

Theorem (Coppersmith)

Given a polynomial f of degree d and N, we can efficiently find all roots r satisfying $f(r) \equiv 0 \bmod N$ when $|r| < N^{1/d}$.

RSA - Stereotyped Messages

- Known most of the message, ex: padding
  - $m = a + x_0,\ x_0 \le R$
  - $c = m^3 = (a + x_0)^3 \bmod n$
- x_0 is a small root of $f(x) = (a + x)^3 - c \pmod n$
- Let the biggest digree of Q be 3
  - $Q(x) = c_3(x^3 + 3ax^2 + 3a^2x + (a^3 - c)) + c_2Nx^2 + c_1Nx + c_0N$
  - $Q(x_0) \le c_3(R^3 + 3aR^2 + 3a^2R + (a^3 - c)) + c_2NR^2 + c_1NR + c_0N$

RSA - Stereotyped Messages (cont.)

- Construct lattice basis

$$\begin{bmatrix} R^3 & 3aR^2 & 3a^2R & a^3 - c \\ & NR^2 & & \\ & & NR & \\ & & & N \end{bmatrix}$$

  - dim L = 4, det L = N^3R^6
- Ignoring approximation factor, we can solve when
  - $|Q(x_0)| \leq |\mathbf{v}| \leq |\det L|^{1/4} < N$
  - $\Rightarrow (N^3R^6)^{1/4} < N$
  - $\Rightarrow R < N^{1/6}$

Achieving the Coppersmith Bound

- Generate lattice from subset of <f(x), N>^{k}
- Allow higher degree polynomials

$$\begin{bmatrix} R^6 & 6aR^5 & 15a^2R^4 & (20a^3-2c)R^3 & (15a^4-6ac)R^2 & (6a^5-6a^2c)R & a^6-2a^3c+c^2 \\ & R^5N & 3aR^4N & 3a^2R^3N & (a^3-c)R^2N & & \\ & & R^4N & 3aR^3N & 3a^2R^2N & (a^3-c)RN & \\ & & & R^3N & 3aR^2N & 3a^2RN & (a^3-c)N \\ & & & & R^2N^2 & & \\ & & & & & RN^2 & \\ & & & & & & N^2 \end{bmatrix}$$

$(R^{21}N^9)^{1/7} < N^2 \Rightarrow R < N^{5/21}$

RSA - Known High Bits of p

- Known large portion of MSBs of one factor
  - $n = pq$, $p = a + x_0$, known a, $x_0 \leq R$
- x_0 is a small roots of $f(x) = a + x \pmod p$
- Construct $Q(x) = 0 \pmod p$
  - $Q(x) = c_1 x(a + x) + c_2(a + x) + N$
  - $Q(x_0) \leq c_1(R^2 + aR) + c_2(R + a) + N$

Theorem (Howgrave-Graham)

Given degree d polynomial f, integer N, we can find roots r modulo divisors B of N satisfying $f(r) \equiv 0 \bmod B$ for $|B| > N^{\beta}$, when $|r| < N^{\beta^2/d}$

RSA - Known High Bits of p

- Construct lattice basis

$$\begin{bmatrix} R^2 & Ra & \\ & R & a \\ & & N \end{bmatrix}$$

  - dim L = 3, det L = NR^3
- Can find the root when
  - $(NR^3)^{1/3} < p = N^{1/2}$
  - $\Rightarrow R < N^{1/6}$

RSA - Partial Key Recovery

- Can factor given 1/2 bits of p [Coppersmith 96]
- Can factor given 1/4 bits of d [Boneh Durfee Frankel 98]
- Can factor given 1/2 bits of d mod (p-1) [Blömer May 03]

(EC)DSA - Known High Bits of k

- Two singature (r_1, s_1), (r_2, s_2), both use small nonces k
  - $s_1 \equiv k_1^{-1}(h_1 + dr_1) \bmod n$
  - $s_2 \equiv k_2^{-1}(h_2 + dr_2) \bmod n$
- Eliminate the variable d
  - $k_1 - s_1^{-1}s_2r_1r_2^{-1}k_2 + s_1^{-1}r_1h_2r_2^{-1} - s_1^{-1}h_1 \equiv 0 \bmod n$
- Let $t = -s_1^{-1}s_2r_1r_2^{-1}$, $u = s_1^{-1}r_1h_2r_2^{-1} - s_1^{-1}h_1$
  - $k_1 + tk_2 + u \equiv 0 \bmod n$
- We wish to solve k_1 and k_2, both small.
  - Let $|k_1|, |k_2| < K$

(EC)DSA - Known High Bits of k

- Construct lattice basis

$$B = \begin{bmatrix} n & 0 & 0 \\ t & 1 & 0 \\ u & 0 & K \end{bmatrix}$$

- The vector **v** = $(-k_1, k_2, K)$ is in this lattice
    - $(-q, k_2, 1)B = (-k_1, k_2, K)$
- Can find **v** when
    - $K < (nK)^{1/3}$
    - $\Rightarrow K < n^{1/2}$