Selected Areas in Cryptology
Cryptanalysis
Week 1

Marc Stevens

stevens@cwi.nl

https://homepages.cwi.nl/~stevens/mastermath/

About myself

- Permanent Researcher
  Cryptology Group
  Centrum Wiskunde & Informatica
  The National Research Center for Mathematics & Computer Science

- Research Interests:
  - Cryptanalysis in general:
    - Hash functions: e.g. breaking MD5 and SHA-1
    - PQC: Lattice, multivariate, codes

- Post-quantum outreach
  - Co-author "The PQC Migration Handbook" https://tinyurl.com/PQCHandbook2
  - Co-organizer of the "PQC Migration Symposium Series" https://post-quantum.nl

My Part of Selected Areas in Cryptology

- Details of lectures & exercises:
  - https://homepages.cwi.nl/~stevens/mastermath/
- Contact:
  - marc@marc-stevens.nl subject "[MM]"
  - MasterMath ELO system
- Exercises are optional, but very recommended
- Some Challenges
  - Cryptanalysis in practice for fun
  - But may need some programming
  - Lots of comparable challenges on https://cryptohack.org/

This lecture

- Recap
  - The field of cryptology
  - Preliminaries
  - The RSA encryption scheme & Shor
  - Post-quantum cryptology

- Symmetric Encryption
  - One-Time Pad
  - Block Ciphers
  - Hellman's Time-Memory-Tradeoff Attack

Cryptology

Cryptology

Building schemes & Testing security

Cryptography:

- **Provably-secure designs:**
  convert attacker against construction
  to attacker against problem
  assumption: problem is intractable

- **Ad-hoc designs:**
  Mainly for symmetric primitives
  Very fast specialized designs
  assumption: design is secure

- **Real-world implementations**

Cryptanalysis:

- **Study proofs:**
  Flaw in proofs?
  Flaw in security model?

- **Study assumptions:**
  Asymptotic cost => insecure?
  Concrete cost => which key sizes?

- **Study structural weaknesses in design**
  Check many attack techniques
  E.g. Linear/Differential cryptanalysis
  …

- **Study real-world systems:**
  side-channels leaking (key) information

Cryptanalysis

- Cryptanalysis studies the security of cryptographic constructions
  - Just 'secure' is ambiguous: means one or more *security properties:*
    - Secrecy: no private information is learned by others
    - Integrity: information has not been modified by others
    - Authenticity/Non-repudiation: origin cannot be disputed
  - 3 main models
    - *Information-theoretical security* (or perfect security):
      Attacker has infinite computing resources
    - *Asymptotic computational security*:
      Attacker is limited to polynomial-sized computing resources
      against hard problems requiring super-polynomial computing resources
    - *Real world security / concrete security*:
      Attacker is limited to real-world computing resources
      against hard problems requiring beyond feasible computing resources

Generic attacks

- Generic attacks
  - Works against every construction of the same type
  - Do not rely on internal structure
  - *Security level upper-bounded by generic attacks*

- Primitive called secure if the best attacks are generic attacks
  - Just means there is no structural weakness
  - But still need sufficiently high security-level in practice:
    No RSA-512 !

Security-level

- What resources are needed for an attack to break a security property?
- Expressed in bits: 128-bit security = an attack requiring 2^{128} 'operations'

- 128-bit security sounds astronomically large, but better to be safe
  - Once 56-bit security was enough: DES by IBM 1975
    Practical brute-force attacks in 1998: EFF's DES cracker
  - 80-bit security was long thought to be sufficient: SHA-1 & 1024-bit RSA
  - But nowadays: Bitcoin network performs 2^{92} hash operations per year

Some preliminaries

Algorithmic cost

Time complexity

= runtime

= number of unit operations
(unit: e.g. bit operation, cpu instruction, function call)

Memory complexity

= amount of unit storage (unit: e.g. bit, byte, block)

Asymptotic complexity functions

Parameter n (bitlength of the input / security parameter)

We write $f(n) = O(g(n))$ if $|f(n)| \leq M\ g(n)$ for all $n \geq n_0$ (for some M, n_0)
(also the called order of the function, only fastest growing term is relevant)

$$poly(n) \coloneqq \{f(n): \mathbb{R} \to \mathbb{R} \mid f(n) = O(n^d), d \in \mathbb{N}\}$$

(set of all functions that are asymptotically bounded by some polynomial)

$$f, g \in poly(n) \quad \Rightarrow \quad f + g,\ f \cdot g,\ f \circ g \in poly(n)$$

Probabilistic & Polynomial Time

Probabilistic algorithms $A(x)$

Uses random coins, non-deterministic

For fixed input, output has probability distribution

PPT := Probabilistic Polynomial-Time

Notation: $x \xleftarrow{r} \mathcal{X}$ (uniformly) randomly sample from $\mathcal{X}$

$\Pr[event]$ = probability event happens

$E[X]$ = the expected value for random variable X

Cryptographic scheme must asymptotically be

efficient: scheme is PPT

secure: attacks should not be PPT

then for any desired gap factor G (e.g. $\mathrm{G} = 2^{128}$)

there exists a n_0 such that for all security parameters $n \geq n_0$:

runtime of attack $\geq G \times$ runtime of scheme

Attack success probability

Success probability for attacks A

= probability algorithm outputs correct solution $y \in Sol(x)$

$p_{succ}^{A}(x) \coloneqq \Pr[y \leftarrow A(x) \wedge y \in Sol(x)]$

Negligible success probability:

$negl(n) \coloneqq \{f : \mathbb{R} \to \mathbb{R} \mid \forall d \in \mathbb{N} : \lim_{n\to\infty} f(n) \cdot n^d = 0\}$

negligible functions vanish to 0, even when multiplied by a polynomial function

E.g.: key guessing attack

- Simply try R random secret keys of n bits
- Finds correct key with probability $R \cdot 2^{-n}$
- Should be negligible
  - In concrete sense: so unlikely that one can disregard this attack
  - As in asymptotic sense: $R \cdot 2^{-n} \in negl(n)$ if $R \in poly(n)$

RSA & Shor

About Alice & Bob

- Alice and Bob want to communicate securely
- Using an insecure channel with a possible adversary

- Securely meaning:
  - Privacy: the adversary cannot learn what messages Alice and Bob are sending
  - Integrity: the adversary cannot change messages between Alice and Bob
  - Authenticity: Bob can verify that it was Alice who send a message

Encryption

- Encryption scheme:
  - Encryption algorithm E uses key K_A
  - Decryption algorithm D uses key K_B
  - For invalid/modified ciphertexts C' decryption D returns $\bot$
- Symmetric/secret-key encryption:
  - Shared secret key: $K_A = K_B$
- Asymmetric/public-key encryption:
  - K_A Bob's public key so anyone can encrypt messages to Bob
  - K_B Bob's private key so only Bob can decrypt messages

RSA Encryption Scheme

- RSA Key generation:
  - Choose p, q large primes at random, let $n = p \cdot q$
  - Then $|\mathbb{Z}_n^*| = \phi(n) = (p-1)(q-1)$
  - Choose e, d such that: $e \cdot d \equiv 1 \bmod \phi(n)$
  - Private key: (n, d)
  - Public key: (n, e)

- Encryption:
  - $C \leftarrow M^e \bmod n$

- Decryption:
  - $M \leftarrow C^d \bmod n$
  - Note $C^d = (M^e)^d \equiv M^{e \cdot d} \equiv M^{e \cdot d \bmod \phi(n)} \equiv M^1 \bmod n$

RSA Security

- Security depends on the following problems to be hard:
  1. Computing M given n, C
  2. Computing d given n, e
  3. Computing $\phi(n)$ given n
  4. Computing p, q given n (i.e. factoring)
- The RSA assumption assumes problem 1 is hard: No PPT algorithm solving it exists
- Problems 2 & 3 are equivalent to the Factoring problem 4

- Best classical algorithm breaking Factoring:
  - Number Field Sieve (NFS) with cost $O(e^{c \cdot (\log n)^{\frac{1}{3}} \cdot (\log \log n)^{\frac{2}{3}}})$
  - Current record: RSA-250 digits $\approx$ 829 bits
- Best quantum algorithm breaking Factoring:
  - Shor's algorithm is quantum polynomial-time
  - Quantum Fourier Transform is used to find the secret period $\phi(n)$

Post-quantum cryptography

Important Quantum attacks

- Shor's algorithm / hidden order finding algorithm
  - Quantum polynomial-time
  - Breaks RSA
  - Breaks Discrete Log: ECC, DSA

- Grover's algorithm / unstructured search algorithm
  - Quantum exponential time $\sqrt{N}$ for search domain size N
  - Quadratic speed-up over classical search *in theory*
  - Search parallelizes embarrassingly by splitting search domain
  - But quantum search using K quantum computers costs $K\sqrt{N/K} = \sqrt{N\,K}$ in total

- And a few others: Simon's algorithm, Kuperberg's algorithm

Impact Future Quantum Computer

- Impact
  - Symmetric cryptography mostly safe
    - ‘Small’ blockciphers need to be avoided due to Grover’s algorithm
    - Still some debate whether 128-bit keyspace is sufficiently large
    - Some modes need to be avoided due to Simon’s algorithm
    - Classical attacks may be transformed in Quantum attacks, but typically at most a quadratic speed-up
  - RSA & ECC will be insecure once a sufficiently large quantum computer exists
    - Latest estimate BSI: “likely to be available within 16 years”
  - New post-quantum cryptography needed

Impact Future Quantum Computer

- Post-quantum public-key frameworks & assumed hard problems
  - Lattices: SVP, LWE, SIS, …
  - Codes: syndrome-decoding, low-weight codeword finding, …
  - Hash functions: preimage finding, second preimage finding, collision finding
  - Isogenies: isogeny (path) finding
  - Multi-variate: solving multi-variate systems
  - MPC-in-the-head: depends on the choice “inside”: can be symmetric crypto

- New/upcoming PQC Standards
  - NIST: ML-KEM: “Kyber”, primary standard for encryption, lattice-based
  - NIST: ML-DSA: “Dilithium”, primary standard for signatures, lattice-based
  - NIST: FN-DSA: “Falcon”, standard for signatures, lattice-based
  - NIST: SLH-DSA: “SPHINCS+”, standard for signatures, hash-based
  - EU/ISO: Frodo: scheme for encryption, lattice-based
  - EU/ISO: McEliece: scheme for encryption, code-based

Symmetric Encryption

One-Time Pad

Symmetric Encryption Schemes

Send message secretly from sender to receiver
Using pre-shared secret key K (unknown to adversary)

Sender encrypts plaintext P to ciphertext C

- Keyspace $\mathcal{K}$, plaintext space $\mathcal{P}$, ciphertext space $\mathcal{C}$
- Function $E_K : \mathcal{P} \to \mathcal{C}$ for $\mathrm{K} \in \mathcal{K}$
- $C = E_K(P)$

Receiver uses corresponding decryption to obtain plaintext P

- $P = D_K(C)$
- $D_K : \mathcal{C} \to \mathcal{P}$

Goals:

- Correctness: $D_K(E_K(P)) = P$ for all K, P
- Secrecy: without key K
  "no information is learned from C about message P"
  (formalization comes later)

One-Time Pad

One-Time Pad (OTP)

For any $l \in \mathbb{N}$:

$$\mathcal{K}_l = \mathcal{P}_l = \mathcal{C}_l = \{0,1\}^l \approx \mathbb{F}_2^l$$
$$E_K(P) \coloneqq P \oplus K$$
$$D_K(C) \coloneqq C \oplus K$$

Requires K uniformly random selected

Key, and Plain- and ciphertext have equal length

$0 \oplus 0 = 1 \oplus 1 = 0$
$1 \oplus 0 = 0 \oplus 1 = 1$
Addition in $\mathbb{F}_2$

Only encryption method providing perfect secrecy

no statistical correlation between cipher- and plaintext if key is unknown

$\Rightarrow$ no information can be learned even with ∞ computing power

$$\Pr_K[C = P \oplus K] = \Pr_K[K = P \oplus C] = 2^{-l}$$

Given C, every plaintext is equally likely

Given P, every ciphertext is equally likely

OTP Issues

Perfect secrecy, but broken if

1. Key K is not kept secret
2. Key K was not selected uniformly at random from $\mathcal{K}_l$
3. Key K is reused for two messages
   attacker learns: $C_1 \oplus C_2 = P_1 \oplus P_2$

Also malleable!

1. Sender encrypts $P =$ "`I owe you 10$`"
2. Attacker intercepts $C = K \oplus P$

   Let $D =$ "`I owe you 10$`" $\oplus$ "`I owe you 5k$`"
   $=$ "`__________10_`" $\oplus$ "`__________5k_`"

   Attacker doesn't even need to know the actual text, only the position and value of the change
3. Attacker sends $C' = C \oplus D$ to receiver
4. Receiver obtains C' and decrypts:
   $P' = K \oplus C' = P \oplus D =$ "`I owe you 5k$`"

Symmetric Encryption

Block Ciphers

Block Cipher

- Block ciphers work differently from the one-time pad
  - Only encrypts fixed-size blocks as a whole (not per bit)
  - Let security parameter n
  - Key space $\mathcal{K}(n)$ and block space $\mathcal{M}(n)$ (e.g., $\{0,1\}^n$)
  - $Enc: \mathcal{K}(n) \times \mathcal{M}(n) \to \mathcal{M}(n)$ such that
    - $Enc_K: \mathcal{M}(n) \to \mathcal{M}(n)$ is a permutation for all $K \in \mathcal{K}(n)$
    - $Dec_K \coloneqq Enc_K^{-1}$ is efficiently computable

  - Note: n is typically omitted: $\mathcal{K}, \mathcal{M}$

Generic attacks

Generic key recovery attack model

List of plaintext + ciphertext pairs: $(P_1, C_1), (P_2, C_2), \ldots$

How are these pairs chosen?

Known plaintext attack: random plaintexts

Chosen plaintext attack: attacker may choose P_i

…

Generic key recovery attack

1. Query l pairs $(P_1, C_1), \ldots, (P_l, C_l)$
2. Walk over search space $K \in \mathcal{K}$
3. If $C_i = Enc_K(P_i)$ for $i = 1, \ldots, l$ then return K
4. Otherwise, if no such K, return $\perp$

Complexity: $O(|\mathcal{K}|)$

Note: Even if there are l pairs to check in total
The first is very likely to fail and the key candidate dismissed,
so most times we only have to check 1 pair

Generic 1-out-of-L key recovery attack

Assume L users with different keys $K_1, \dots, K_L$
Attacker succeeds if it finds 1 key

Generic attack

1. Chooses l plaintexts $P_1, \dots, P_l$
2. Queries encryptions for each user:
   $C_{i,j} = Enc_{K_j}(P_i)$ for $i = 1, \dots, l$ and $j = 1, \dots, L$
3. Walks over search space $K \in \mathcal{K}$
4. Compute $\tilde{C}_1 = Enc_K(P_1)$
5. For j such that $C_{1,j} = \tilde{C}_1$ do
6. If $C_{i,j} = Enc_K(P_i)$ for $i = 2, \dots, l$ then return K
7. Otherwise, return $\perp$

Every key guess has success probability $L/|\mathcal{K}|$
Complexity: $O(|\mathcal{K}|/L)$
Speed up by factor L!

Attacks with precomputation

There are attacks that cost $O(|\mathcal{K}|)$ or more in total, but < $O(|K|)$ per problem instance

Two phases:

An offline part that performs at least $O(|\mathcal{K}|)$ operations

An online part that attacks each of the L keys independently

An extreme example, codebook dictionary:

Offline: 1. Choose block B
2. Create hash table with $(Enc_K(B), K)$ entries

Time: O($|\mathcal{K}|$), Memory: O($|\mathcal{K}|$)

Online: 1. For each secret key K_i to be attacked
2. Query $C = Enc_{K_i}(B)$
3. Find table entry (C, K_i)

Time: $O(1)$, Memory: $O(|\mathcal{K}|)$

Non-uniform attacks:

Make pre-computed data part of online attack algorithm

Online algorithm now only has total cost $< O(|\mathcal{K}|)$

Hellman's
Time-Memory trade off attack

Hellman's Time-Memory trade off attack

An attack that uses more time, but less memory

Idea:

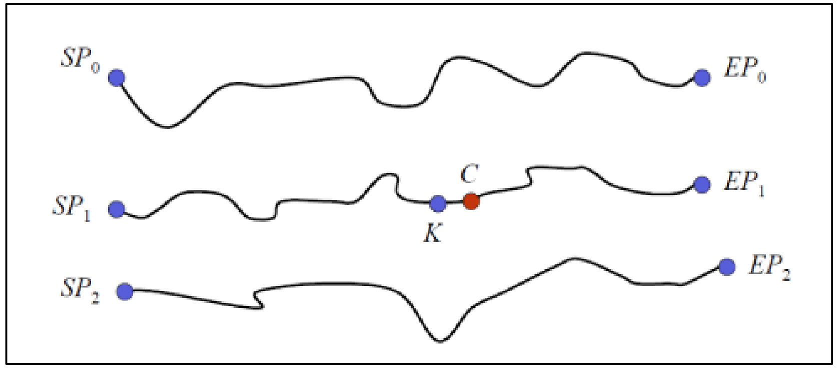
Use fixed block B and map $\phi: \mathcal{C} \to \mathcal{K}$

Iterative function $F: \mathcal{K} \to \mathcal{K}$ 'walks' through key space

$F(K_i) = K_{i+1}$, where $C = Enc_{K_i}(B)$, $K_{i+1} = \phi(C)$

Offline: Store many long walks covering key space

Only store begin and endpoints $(SP_j, EP_j = F^t(SP_j))$



Online: *Query* $C_0 = Enc_K(B)$, *compute* $K_0 = \phi(C_0)$ (Hence by definition: $F(K) = K_0$)

Compute walk from K_0 until say endpoint EP_1 is found

Find secret K by walking from SP_1

Hellman's Time-Memory trade off attack

Setup Details:

$F: \mathcal{K} \to \mathcal{K}$, where $F = \phi \circ E$

$E: \mathcal{K} \to \mathcal{M}$, $\quad E(K) \coloneqq Enc_K(B)$

$\phi: \mathcal{M} \to \mathcal{K}$ needs to be surjective

If $|\mathcal{M}| \geq |\mathcal{K}|$ then easy, otherwise impossible

When $|\mathcal{M}| < |\mathcal{K}|$

Use multiple blocks $B_1, B_2, \ldots, B_l$ such that $|\mathcal{M}|^l \geq |\mathcal{K}|$

$E: \mathcal{K} \to \mathcal{M}^l$, $\quad E(K) \coloneqq \left(Enc_K(B_1), \ldots, Enc_K(B_l)\right)$

And surjective map $\phi: \mathcal{M}^l \to \mathcal{K}$

Hellman's Time-Memory trade off attack

Simplified version

Attack parameters: Number of walks: m
Length of each walk: t

Offline attack:
1. Choose $SP_1, \dots, SP_m$ uniformly at random from $\mathcal{K}$
2. Compute $EP_i = F^t(SP_i)$ for $i = 1, \dots, m$
3. Store (EP_i, SP_i) in hash table / sorted table

Online attack:
1. Given $C_0 = Enc_K(B)$ for some unknown key K
2. Let $P_0 = \phi(C_0)$
3. For $i = 0, \dots, t-1$
4. If $P_i = EP_j$ for some j then
5. Let $\widetilde{K} := F^{t-i-1}(SP_j)$
6. If $Enc_{\widetilde{K}}(B) = C_0$ then return $\widetilde{K}$
7. Compute $P_{i+1} := F(P_i)$
8. Otherwise, return $\perp$

Hellman's Time-Memory trade off attack

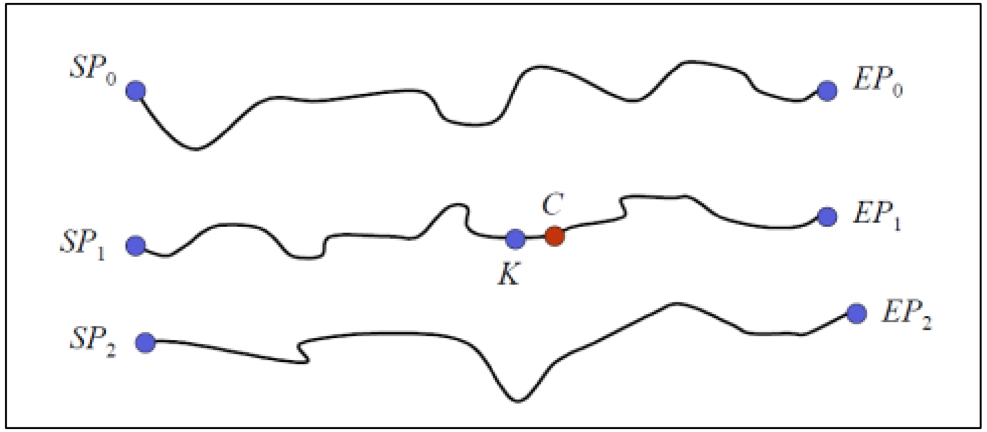
Simplified version analysis

Ideally, use $m \cdot t = |\mathcal{K}|$ and
hope to cover entire space

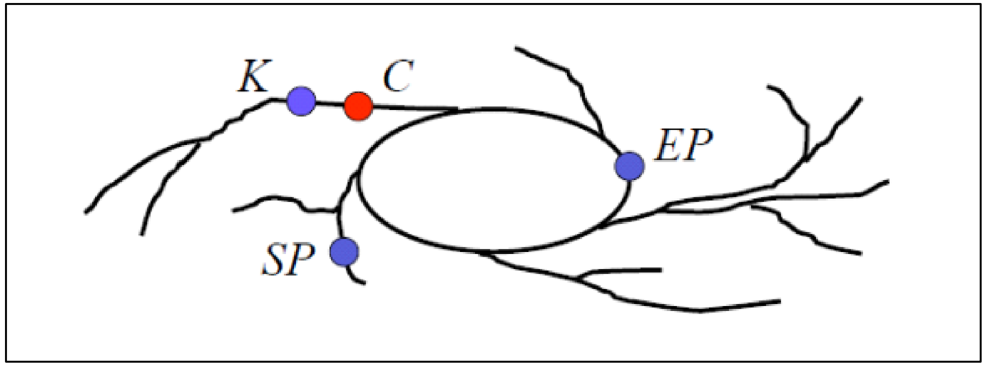
However, F behaves as a random function

Thus many collisions $F(x) = F(y)$ exist and merges walks

- Substantial part of space is never reached
- Creates false alarms:
  - K does not actually lie on walk from SP_i, but on walk from another SP with same EP_i



Ideal situation



Expected situation for random functions

Hellman's Time-Memory trade off attack

Simplified version analysis

- Collisions start to occur when $m \cdot t \approx \sqrt{|\mathcal{K}|}$
- Due to the birthday paradox (covered later)
- The expected number of collisions grows roughly quadratic in $m \cdot t$
- False alarms analysis:
  - Walk from P_0 has t points
  - There are at most $m \cdot t$ points covered by the table
  - Each pair has probability $1/|\mathcal{K}|$ to collide and cause false alarm (i.e. without P_0 actually being on the walk)
  - Expected number of false alarms: $E[Z] \leq m \cdot t^2/|\mathcal{K}|$
  - Expected costs of false alarm: $t \cdot E[Z] \leq m \cdot t^3/|\mathcal{K}|$

- Success only if target K is covered (part of a walk from a SP_i)
- Hellman: when $m \cdot t^2 = |\mathcal{K}|$, success probability is $\approx 0.80mt/|\mathcal{K}|$

Hellman's Time-Memory trade off attack

Improved version

Use r independent tables with different $\phi_1, \dots, \phi_r$

Even if the same key is covered in different tables
then different ϕ_i imply different walks instead of merging walks

Hellman proposed $m = t = r = \sqrt[3]{|\mathcal{K}|}$

Individual table: success probability $\approx 0.80 \;/\; \sqrt[3]{|\mathcal{K}|}$

Total success probability ≈ 0.8

Offline time complexity: $O(mtr) = O(|\mathcal{K}|)$

Offline memory complexity: $O(rm) = O(|\mathcal{K}|^{2/3})$

Online complexity:

$$O\left(rt + rmt^3/|\mathcal{K}|\right) = O\left(|\mathcal{K}|^{2/3} + |\mathcal{K}|^{2/3}\right) = O(|\mathcal{K}|^{2/3})$$