HW 1 Answers

CSCI-UA.0480-63

September 30, 2025

1. Threat modeling for Citi Bike

We propose three security policies for New York's Citi Bike system, each targeting a distinct adversary type (thieves, terrorists, trolls). For each policy, we specify concrete enforcement mechanisms and rationale.

Policy A (Thieves): Ensure bikes cannot be used or resold if unlawfully removed

Goal: Reduce economic incentive by making stolen bikes valueless and hard to monetize.
Mechanisms:

- *Tamper-resistant electronic locks with authenticated release*: Require station or on-bike lock to perform mutual authentication with backend before unlock; deny offline unlocks; sign unlock events to enable audit. Implement geo-fencing so a bike that leaves a service area without an authorized session hard-locks. (Defense-in-depth policy/mechanism separation discussed in Notes 2025.01.01.)
- *On-bike immobilizer with cryptographic pairing*: Pair controller to per-bike keys; if controller or battery is replaced without re-provisioning via service tool, firmware refuses to engage motor/gears. (Avoid security by obscurity; rely on secret keys rather than hidden design; cf. Kerckhoffs principle, Notes 2025.01.04.)
- *Active recovery and deterrence*: GPS/Cellular beaconing with periodic attestations; stolen-mode triggers brighter lighting patterns and server-side location beacons for recovery in collaboration with NYPD.

Why it works: Removes resale utility and increases recovery likelihood, shifting attacker utility (Notes 2025.01.01 on attacker/defender utilities).

Policy B (Terrorists): Maintain rider safety and rapid incident response

Goal: Preserve safety and availability in the face of attempts to cause violent disruption.
Mechanisms:

- *Station and fleet integrity monitoring*: Backend analytics for simultaneous abnormal unlocks, sudden mass rebalancing anomalies, or tamper alerts indicating coordinated sabotage; automated incident playbooks to disable unlocks in affected zones.
- *Physical hardening and surveillance at high-risk sites*: Bollards, tamper-evident fasteners, and CCTV coverage at major transit hubs to deter placement of hazards and to support forensics.

- *Emergency broadcast and geo-fenced shutdown*: Capability to pause rentals and push safety notices within a radius; integrate with city emergency systems for coordinated response.

Why it works: Defense-in-depth across detection, prevention, and response reduces likelihood and impact (Notes 2025.01.01 on defense-in-depth).

Policy C (Trolls): Maintain service availability and fair access

Goal: Limit nuisance attacks (dock blocking, false reports, mass reservation griefing).
Mechanisms:

- *Strong user authentication and rate-limiting*: Bind accounts to verified payment tokens; apply per-user and per-IP quotas on reservations, unlock attempts, and incident reports; require proof-of-ride (e.g., short post-unlock movement) to keep a dock reserved.
- *Anomaly scoring and graduated friction*: Add friction (CAPTCHAs, SMS recheck) when behavior deviates from normal usage to deter automated griefing.
- *Dock occupancy attestation*: Use on-dock sensors plus bike IMU to verify a dock is truly occupied; auto-release ghost holds.

Why it works: Increases attacker cost for nuisance actions while minimizing friction for legitimate riders (Notes 2025.01.01 on policy vs. mechanism and attacker utilities).

References to provided Notes

- Notes 2025.01.01: Introduction & threat modeling; policy vs. mechanism; defense-in-depth; attacker/defender utilities. Example passages used for rationale.
- Notes 2025.01.04: Kerckhoffs's principle and avoiding security by obscurity; adversary models.

Selected excerpts (for traceability)

- Security Text 2025.01.01.txt, around line 1:
  Chapter 1: Introduction & threat modeling Suggested reading: • Security Engineering, Chapter 1
- Security Text 2025.01.01.txt, around line 3:
  • Security Engineering, Chapter 1 • Tools & Jewels Chapter 1 • An introduction to approachable threat modeling • Attack trees Case study:
- Security Text 2025.01.01.txt, around line 10:
  Security is fundamentally a contest between an attacker and defender. Defenders have security policies they wish to enforce and attackers, for various reasons, aim to undermine those policies.

  Security engineering is about designing mechanisms for defenders to prevent attackers from causing harm. Defenders also must balance competing requirements such as a budget, ease-of-use and compliance with the law.

  The most challenging aspect of security engineering is that attackers adapt and evolve over time. Security engineers continually try to predict what attackers will do in response to a new system being deployed so they can build in security in advance.
- Security Text 2025.01.01.txt, around line 45:
  The distinction between the two notions is not binary. Some security measures provide both. For example, locking your door when you arrive home provides some feeling of protection from the outside world. It also actually makes it more difficult for potential intruders to get inside (though in most cases probably far less than people believe). Deceptive security mechanisms: security theater and deterrents

While almost everybody understands rationally that sleeping under a blanket provides no real security, sometimes elaborate security mechanisms are designed to appear effective to the general public although security engineers know they have little effect. This is called security theater (as coined by Bruce Schneier) or sometimes placebo security. Airport security is often cited as an example of security theater. An infamous example was the use of “puffer machines” (ETPs) in the US in the mid-2000s. They were consistently found to have no effectiveness at bomb detection under real operating conditions. It’s been speculated they were kept in operation for years because the high-tech-looking machines helped ease public fear about flying after September 11th.

- Security Text 2025.01.01.txt, around line 47:
  Deceptive security mechanisms: security theater and deterrents

  While almost everybody understands rationally that sleeping under a blanket provides no real security, sometimes elaborate security mechanisms are designed to appear effective to the general public although security engineers know they have little effect. This is called security theater (as coined by Bruce Schneier) or sometimes placebo security. Airport security is often cited as an example of security theater. An infamous example was the use of “puffer machines” (ETPs) in the US in the mid-2000s. They were consistently found to have no effectiveness at bomb detection under real operating conditions. It’s been speculated they were kept in operation for years because the high-tech-looking machines helped ease public fear about flying after September 11th.

  Deception about the effectiveness of a security mechanism can also be useful if potential attackers don’t understand the true effectiveness and are scared off. This is called a deterrent effect. A common example is shops which mount fake security cameras to deter shoplifters. Or more classically, scarecrows or plastic owls can scare away unwanted birds.

- Security Text 2025.01.01.txt, around line 49:
  While almost everybody understands rationally that sleeping under a blanket provides no real security, sometimes elaborate security mechanisms are designed to appear effective to the general public although security engineers know they have little effect. This is called security theater (as coined by Bruce Schneier) or sometimes placebo security. Airport security is often cited as an example of security theater. An infamous example was the use of “puffer machines” (ETPs) in the US in the mid-2000s. They were consistently found to have no effectiveness at bomb detection under real operating conditions. It’s been speculated they were kept in operation for years because the high-tech-looking machines helped ease public fear about flying after September 11th.

  Deception about the effectiveness of a security mechanism can also be useful if potential attackers don’t understand the true effectiveness and are scared off. This is called a deterrent effect. A common example is shops which mount fake security cameras to deter shoplifters. Or more classically, scarecrows or plastic owls can scare away unwanted birds.

  Finally, the security industry is also littered with snake oil: products deceptively claiming to provide security which in fact are useless. These may fool the owner, but not potential attackers. The ADE 651 machine was widely sold in Iraq, though it literally did nothing except look like a bomb detector. Many argue that anti-virus software in the modern era is largely snake oil. Snake oil might offer some psychological benefit as security theater. The distinction is that with security theater the operator knows the mechanism is ineffective; with snake oil they’ve been deceived.

2. Hash functions and privacy

Prompt (abridged): BobCrypt uploads hashes of phone numbers to discover friends while claiming one-wayness preserves privacy.

(a) Why plain hashing phone numbers does not protect privacy

Phone numbers live in a tiny, structured space (e.g., 10–15 digits with known national formats). An adversary can simply enumerate all plausible phone numbers, hash each, and match the server’s values. This defeats confidentiality despite hash one-wayness (one-wayness resists inverting a random preimage, not exhaustive enumeration of low-entropy domains). It also leaks users' entire address books to the server via set-membership queries.

Citations from Notes - Notes 2025.01.03: hash properties (collision, second-preimage), random-oracle intuition; one-wayness does not stop brute-force over small domains. - Notes 2025.02.02: hash usage in protocols and pitfalls when metadata or side channels leak information.

(b) Two mitigations and trade-offs

Mitigation 1: Per-user salting (H(phone — user_salt)) with keyed hashing for queries

- Server stores for each account a keyed digest, e.g., HMAC_k(phone) with server-held key k. Client fetches a short-lived token (MAC key or PRF token) and computes HMAC_k(address-book entries) locally to query. Without k, bulk offline rainbow/dictionary attacks on leaked hashes are infeasible; with k rotated, tokens limit abuse.
- *Trade-offs*: Server learns query results and could still enumerate if it misuses k. Requires online token issuance and rate-limiting.

Mitigation 2: Privacy-preserving contact discovery (k-anonymity bucketization or PSI)

- *K-anonymous buckets*: Client sends first r bits of H(phone) to fetch a bucket of size $\geq k$; performs local filter with full H(phone). Server sees only bucket queries, not exact hashes.
- *Private Set Intersection (PSI)*: Use a PSI protocol so client learns intersection of its address book with server's registered numbers, and server learns nothing about the rest. Efficient variants use OPRF/PRF-based PSI with rate limits.
- *Trade-offs*: K-anon leaks coarse prefix information and allows frequency attacks; PSI adds engineering and compute cost but offers strongest privacy.

(c) A better baseline design

Combine defense-in-depth:

- Use an OPRF/PSI-based contact discovery service as the default.
- If PSI unavailable, fall back to k-anonymous buckets with per-session OPRF tokens and strict rate limits.
- Never store raw hashes of phone numbers; store HMAC_k(phone) server-side with periodic key rotation and audit.

This aligns with the principle to avoid security by obscurity and rely on secret keys and provable primitives.

Selected excerpts (for traceability)

- Security Text 2025.01.03.txt, around line 1:
  Chapter 3: Authenticated storage, Hash functions and MACs Suggested reading: • Security Engineering Chapters 5.3.1, 5.6
- Security Text 2025.01.03.txt, around line 3:
  • Security Engineering Chapters 5.3.1, 5.6 Advanced reading: • GCAC 8 (Hash functions) Bonus reading: • The first SHA-1 collision (2017)
- Security Text 2025.01.03.txt, around line 5:
  • GCAC 8 (Hash functions) Bonus reading: • The first SHA-1 collision (2017) • More practical SHA-1 collisions (2019) • SHA-256 visualization
- Security Text 2025.01.03.txt, around line 6:
  Bonus reading: • The first SHA-1 collision (2017) • More practical SHA-1 collisions (2019) • SHA-256 visualization • SHA-3 visualization
- Security Text 2025.01.03.txt, around line 10:
  • SHA-3 visualization

  One more essential primitive to introduce is the cryptographic hash function. It's perhaps the most widely used primitive in all of cryptography. Hash functions are the workhorse underlying almost every protocol from encryption to cryptocurrency.

  We'll start with a simple application, a client uploading a large file to a cloud server:
- Security Text 2025.01.03.txt, around line 22:
  The client doesn't care about keeping the file secret, but wants to be sure that later when it downloads the file, it receives the same file f'=f. Obviously the client could just store the entire original file, but then why use a cloud server?

  Instead, we'll have the client store the hash of the file H(f), and then later check if H(f)=H(f'). Hash functions are similar to PRFs in that the output looks somewhat random. Hash functions only take one input, not two and that input can be infinitely long. That's right: hash functions are defined as a function H: {0,1}* → {0,1}t, they take any number of bits as input and then produce a t-bit output, sometimes called a digest.

  Our file-storage protocol will be secure as long as the server can't find any other file with the same hash. This property is called second-preimage-resistance and is defined with the following game:
- Security Text 2025.01.03.txt, around line 24:
  Instead, we'll have the client store the hash of the file H(f), and then later check if H(f)=H(f'). Hash functions are similar to PRFs in that the output looks somewhat random. Hash functions only take one input, not two and that input can be infinitely long. That's right: hash functions are defined as a function H: {0,1}* → {0,1}t, they take any number of bits as input and then produce a t-bit output, sometimes called a digest.

  Our file-storage protocol will be secure as long as the server can't find any other file with the same hash. This property is called second-preimage-resistance and is defined with the following game:

  • The challenger randomly chooses x and sends it to the adversary.
- Security Text 2025.01.03.txt, around line 32:
  You should be able to convince yourself that if this property is true, the above file storage protocol is secure. This approach is widely used in practice. For example, official releases of software packages like Ubuntu Linux often publish the hash of the authentic version of the software. If you obtain the hash of the desired file securely, you can then download the (often quite large) file from any untrusted service like BitTorrent and be sure you've obtained the correct version by checking its hash.

  Cryptographic hash functions are also designed to be one-way, but the definition is quite similar to that for one-way functions as above so we'll skip it here. Hash functions and collision-resistance

3. MACs vs. PRFs

(a) PRF implies MAC (reduction)

Assume there exists an adversary A_1 that wins the MAC existential unforgeability game against F with non-negligible probability. We construct A_2 to win the PRF distinguishing game using A_1 as a subroutine:

- A_2 receives oracle $O(\cdot)$ which is either $F(k,\cdot)$ or a truly random function $R(\cdot)$.
- A_2 runs A_1 and answers each MAC query m by returning $t = O(m)$.
- When A_1 outputs a purported forgery (m^*, t^*) with m^* not previously queried, A_2 outputs "real" iff $O(m^*)=t^*$, else "random".

Analysis: If $O=F(k,\cdot)$ then A_1 sees a perfect MAC oracle and, by assumption, produces a valid new (m^*, t^*) with non-negligible probability, so A_2 distinguishes PRF-from-random with the same advantage. If $O=R(\cdot)$, then t^* is independent of $R(m^*)$ and matches with probability at most 2^{-t} (negligible). Hence any MAC forger yields a PRF distinguisher, proving a secure PRF is also a secure MAC.

(b) A MAC that is not a PRF

Let F be a secure PRF. Define F'(k, m) that outputs the pair $(F(k, m), 0^t)$ (or equivalently, append a fixed trailer bit). Verification accepts a tag t' iff its first t bits equal F(k, m). Then:

- **MAC security**: Given oracle access, producing a fresh (m, t') requires predicting F(k, m) on a new m, which is hard if F is a PRF. The fixed trailer does not help forgery.
- **Not a PRF**: F' is trivially distinguishable from random because its output always ends with 0^t. A random function produces that pattern with probability 2^{-t}, yielding a huge distinguishing advantage.

Citations to provided Notes

- Notes 2025.01.02: PRF definition and game (indistinguishability from random). - Notes 2025.01.03 and 2025.01.05: MAC definition and EUF-CMA game; AE and key separation context.

Selected excerpts (for traceability)

- Security Text 2025.01.02.txt, around line 1:
  Chapter 2: Identification protocols, One-way functions, and PRFs Suggested reading: • Security Engineering Chapters 3.1, 3.2, 3.3
- Security Text 2025.01.02.txt, around line 5:
  Advanced reading. Note: In this class we will skip a lot of mathematical formalism around crypto. If you want to read about these concepts with full mathematical rigor, check out A Graduate Course in Applied Cryptography by Dan Boneh and Victor Shoup. • GCAC 18.6 (challenge-response protocols) • GCAC 4.4 (PRFs) Bonus reading: • Anatomy of a Subway Hack
- Security Text 2025.01.02.txt, around line 112:
  Returning to the door example, a one-way function is actually not enough! It will prevent an adversary from guessing the key, but the adversary doesn't need to guess the key to open the door. They only need to guess a correct response to the door's challenge. Pseudorandom functions (PRFs) Let's go back to the second property: the adversary shouldn't be able to guess the output of the function. This requires a stronger property than the above. We'll define a new game for it:

- Security Text 2025.01.02.txt, around line 117:
  • The challenger generates a random bit b. If b is 1, they generate a random key k. • The adversary repeatedly chooses a nonce n and sends it to the challenger. • If b=1, the challenger responds with F(k, n), honestly computing the PRF on the adversary's query n. Assume this output is
  lambda bits long. Otherwise, if b=0, the challenger responds with
  lambda randomly generated bits. • The adversary outputs a guess b'. • If b is equal to b', the adversary wins.

- Security Text 2025.01.02.txt, around line 121:
  • If b is equal to b', the adversary wins.

  In this game, the challenger picks a random bit b to decide if the adversary sees real or random output. We'll still assume the adversary is adaptive and PPT. But now they can win the game half the time by random guessing, so we'll need to change our security definition to say that F is a PRF if no PPT adversary can win the game with probability greater than 0.5 + negl(
  lambda). We call this a negligible advantage over the baseline probability of 0.5 of randomly guessing b correctly.

  A function F that satisfies this definition is secure enough for the door-opening protocol. In fact, a careful reader might notice it's actually stronger than we need it to be: this definition states that an adversary can't distinguish the function's output from random, but all we need is that they can't predict the function's output exactly. We'll eventually design the precise primitive we need (a MAC function) in the coming chapters.

- Security Text 2025.01.02.txt, around line 123:
  In this game, the challenger picks a random bit b to decide if the adversary sees real or random output. We'll still assume the adversary is adaptive and PPT. But now they can win the game half the time by random guessing, so we'll need to change our security definition to say that F is a PRF if no PPT adversary can win the game with probability greater than 0.5 + negl(
  lambda). We call this a negligible advantage over the baseline probability of 0.5 of randomly guessing b correctly.

  A function F that satisfies this definition is secure enough for the door-opening protocol. In fact, a careful reader might notice it's actually stronger than we need it to be: this definition states that an adversary can't distinguish the function's output from random, but all we need is that they can't predict the function's output exactly. We'll eventually design the precise primitive we need (a MAC function) in the coming chapters.

  Also note that the one-way property is necessary, but not sufficient, for building a PRF. If an adversary could win the OWF game, they could win the PRF game as well (first solve for the key, then use that knowledge to distinguish the output of the PRF). We'll talk much more about how to reason about the relationship between security properties.

- Security Text 2025.01.02.txt, around line 125:
  A function F that satisfies this definition is secure enough for the door-opening protocol. In fact, a careful reader might notice it's actually stronger than we need it to be: this definition states that an adversary can't distinguish the function's output from random, but all we need is that they can't predict the function's output exactly. We'll eventually design the precise primitive we need (a MAC function) in the coming chapters.

  Also note that the one-way property is necessary, but not sufficient, for building a PRF. If an adversary could win the OWF game, they could win the PRF game as well (first solve for the key, then use that knowledge to distinguish the output of the PRF). We'll talk much more about how to reason about the relationship between security properties. Random oracles and ideal functionalities We'll talk more about how to actually build a PRF later. For now, just know that we can build PRFs that are suitable for a secure challenge-response protocol.

- Security Text 2025.01.02.txt, around line 127:
  Also note that the one-way property is necessary, but not sufficient, for building a PRF. If an adversary could win the OWF game, they could win the PRF game as well (first solve for the key, then use that

knowledge to distinguish the output of the PRF). We'll talk much more about how to reason about the relationship between security properties. Random oracles and ideal functionalities We'll talk more about how to actually build a PRF later. For now, just know that we can build PRFs that are suitable for a secure challenge-response protocol.

The name "pseudorandom function" is telling. We want a function that is "almost" a random function. Why not just use a random function? We could design a truly random function by generating and storing a function table (the mapping from every input to every output) completely at random. But storing this function table would require an impractical amount of memory. For example, consider a random 2-bit function:

4. Semantic security for the Vigenère cipher

The Vigenère cipher with a fixed period p=10 is not semantically secure. We give a no-query adversary that wins the IND-CPA game with probability 1.

Adversary strategy

1. Choose two equal-length plaintexts m0, m1 longer than 10 so that for some position i, the letters at all indices congruent to i mod 10 differ between m0 and m1 but are identical within each plaintext across those positions. For example, for English letters let the positions i, i+10, i+20, ... in m0 all be 'A' and in m1 all be 'B'; keep all other positions equal between m0 and m1.

2. Submit m0, m1 as the challenge messages. Let the challenger encrypt mb under a uniformly random 10-letter key k and return c.

3. Inspect ciphertext letters at positions i, i+10, i+20, ...: under Vigenère, each of these positions is a shift by the same key letter k[i]. Because the plaintext letters at those positions are constant within a message, the corresponding ciphertext letters will also be constant (but shifted) within each message. Compute the shift between c[i] and c[i+10]. If they are equal shifts for 'A', output b=0; if they align to the shift for 'B', output b=1.

This succeeds with probability 1 since the repeated key letter induces identical shifts at every i+10j position, letting the adversary tell which constant letter ('A' vs 'B') was encrypted.

Why this breaks semantic security

IND-CPA requires that no efficient adversary can distinguish encryptions of chosen equal-length messages with probability non-negligibly greater than 1/2. Here, the periodic key leaks class information (the congruence class modulo 10), enabling frequency/structure-based distinguishing on repeated positions, violating semantic security. The weakness is exactly the key-reuse periodicity. In contrast, a one-time pad (non-repeating key) would be semantically secure, and modern randomized modes ensure semantic security by using nonces/IVs.

Citations to provided Notes

- Notes 2025.01.04: description and break of Vigenère via guessing period and frequency on residue classes; one-time pad as non-repeating key. - Notes 2025.01.05: formalization of semantic security and why deterministic/structure-leaking modes fail.

Selected excerpts (for traceability)

- Security Text 2025.01.04.txt, around line 7:
  • The Code Book by Simon Singh (no free copy online) provides an excellent history of cryptography and early ciphers. • Playfair ciphers, including the famous message encrypted (by hand) by future president John F. Kennedy. • Historical uses of one-time pads, from the Crypto Museum • A critique of one-time pads in the modern setting • Successful attacks on RC4 in practice which led to the cipher's deprecation
- Security Text 2025.01.04.txt, around line 8:
  • Playfair ciphers, including the famous message encrypted (by hand) by future president John F. Kennedy. • Historical uses of one-time pads, from the Crypto Museum • A critique of one-time pads in the modern setting • Successful attacks on RC4 in practice which led to the cipher's deprecation
- Security Text 2025.01.04.txt, around line 72:
  You might try to fix this by making the mapping of plaintext letters to ciphertext letters more complicated, say by choosing a random permutation of the alphabet as the key. There are now 26! possible keys, or roughly 288. This is arguably large enough that even modern computers cannot guess all possibilities (or at least not without extreme cost).

  But there's a bigger problem with any monoalphabetic cipher like this which always encrypts a given letter in plaintext to a fixed ciphertext letter. An attacker can perform frequency analysis. In English, "E" is the most common letter, so the most common letter appearing in ciphertext is probably the encryption of E (given enough ciphertext for the law of large numbers to kick in). The distribution of letters is generally uneven:
- Security Text 2025.01.04.txt, around line 79:
  Monoalphabetic ciphers often appear in newspapers as a puzzle to solve. Even with only a few sentences it's usually breakable on paper. Computers can quickly do it.

  Early patches to this scheme involved tricks like using more than 26 symbols in the ciphertext alphabet and randomly mapping a common letter like "E" to one of several possibilities to even out the frequency of ciphertext letters. Special ciphertext characters can be used to mean "repeat the last letter" or "delete the previous letter." Versions of this have been re-invented many times, such as by the Zodiac serial killer who invented a cipher using 340 characters.

  While monoalphabetic ciphers are too weak to be used for any critical purposes, they live on in Internet forums in the form of rot13, which is simply a cipher shifting all letters by 13 places in the alphabet (notice that this means decryption is the same as encryption). This is used to scramble online forum posts to avoid spoilers.
- Security Text 2025.01.04.txt, around line 82:
  While monoalphabetic ciphers are too weak to be used for any critical purposes, they live on in Internet forums in the form of rot13, which is simply a cipher shifting all letters by 13 places in the alphabet (notice that this means decryption is the same as encryption). This is used to scramble online forum posts to avoid spoilers. Historical encryption: polyalphabetic ciphers Eventually, more complicated polyalphabetic ciphers were developed where a character in plaintext does not always map to the same character in ciphertext. For example, with the 15th century Vigenère cipher, p different monoalphabetic ciphers are rotated, the ith letter of ciphertext is encrypted using key[i mod p] where p is the length of the key or period. Here is an example with a key "CAT" with p=3:
- Security Text 2025.01.04.txt, around line 83:
  While monoalphabetic ciphers are too weak to be used for any critical purposes, they live on in Internet forums in the form of rot13, which is simply a cipher shifting all letters by 13 places in the alphabet (notice that this means decryption is the same as encryption). This is used to scramble online forum posts to avoid spoilers. Historical encryption: polyalphabetic ciphers Eventually, more complicated polyalphabetic ciphers were developed where a character in plaintext does not always map to the same character in ciphertext. For example, with the 15th century Vigenère cipher, p different monoalphabetic ciphers are rotated, the ith letter of ciphertext is encrypted using key[i mod p] where p is the length of the key or period. Here is an example with a key "CAT" with p=3:

- Security Text 2025.01.04.txt, around line 92:
  At first glance, this is much more difficult to break as simple frequency analysis won't work. Indeed for a period of hundreds of years the best cryptanalysts in the world apparently didn't know how to attack this cipher and was nicknamed "le chiffre indéchiffrable" (the unbreakable cipher). It wasn't until the 19th century that cryptanalysts (among them Charles Babbage) noticed a relatively simple attack.

  This cipher can be broken by first guessing the period p, then performing frequency analysis on subsets of letters. For example, if an analyst correctly guesses that p=3 in the example above, then they know the first letter, fourth letter, seventh letter and so on will all be encrypted using the same character of the key. They can repeat this frequency analysis for the second letter, fifth letter etc. and in general for p subsets of letters. This is tedious enough to do by hand that it isn't a popular newspaper puzzle, but it's easy enough to make this approach insecure even without computers.

- Security Text 2025.01.04.txt, around line 94:
  At first glance, this is much more difficult to break as simple frequency analysis won't work. Indeed for a period of hundreds of years the best cryptanalysts in the world apparently didn't know how to attack this cipher and was nicknamed "le chiffre indéchiffrable" (the unbreakable cipher). It wasn't until the 19th century that cryptanalysts (among them Charles Babbage) noticed a relatively simple attack.

  This cipher can be broken by first guessing the period p, then performing frequency analysis on subsets of letters. For example, if an analyst correctly guesses that p=3 in the example above, then they know the first letter, fourth letter, seventh letter and so on will all be encrypted using the same character of the key. They can repeat this frequency analysis for the second letter, fifth letter etc. and in general for p subsets of letters. This is tedious enough to do by hand that it isn't a popular newspaper puzzle, but it's easy enough to make this approach insecure even without computers.

  To address this weakness, the autokey cipher uses a new key to encrypt every letter of plaintext. The sequence of letters used is called the keystream. The first p letters of keystream are the key k itself. After that, keystream letters are previous plaintext letters:

5. Block cipher modes of operation: plaintext block chaining (PBC)

Let PBC be defined by $c_0 = E_K(m_0) \oplus IV$ and for $i \geq 1$: $c_i = E_K(m_i) \oplus m_{i-1}$. This lets the sender evaluate all $E_K(m_i)$ in parallel, then xor with the known previous plaintext (or IV for the first block).

Attack when $m_1 = m_2 = x$ (known)

Given public IV, c_0, c_1, c_2 and a known block x with $m_1 = m_2 = x$: 1) From $c_2 = E_K(m_2) \oplus m_1 = E_K(x) \oplus x$, compute $E_K(x) = c_2 \oplus x$.

2) From $c_1 = E_K(m_1) \oplus m_0 = E_K(x) \oplus m_0$, recover the first block:

$$m_0 = E_K(x) \oplus c_1 = (c_2 \oplus x) \oplus c_1.$$

This uses only public values and the known x. Hence PBC is not semantically secure: repetitions of plaintext blocks leak $E_K(x)$ and reveal neighboring plaintexts.

Citations to provided Notes

- Notes 2025.01.05: modes of operation (ECB/CBC/CTR), IV requirements, and semantic security; parallelizable CTR vs serial CBC. - Notes 2025.01.04: IV/nonce reuse hazards and stream-cipher viewpoint.

Selected excerpts (for traceability)

- Security Text 2025.01.05.txt, around line 8:
  Bonus reading: • EFF DES Cracking project • The ECB Penguin • The POODLE attack exploiting CBC mode with incorrect error handling • The Stick Figure Guide to AES
- Security Text 2025.01.05.txt, around line 9:
  • EFF DES Cracking project • The ECB Penguin • The POODLE attack exploiting CBC mode with incorrect error handling • The Stick Figure Guide to AES
- Security Text 2025.01.05.txt, around line 12:
  • The Stick Figure Guide to AES

  Stream ciphers are an effective means of ensuring confidentiality of data. Though we noted that RC4 is no longer secure and most modern stream ciphers are actually built out of block ciphers, which are among the most essential of crypto primitives. While stream ciphers work by outputting an unbounded stream of pseudorandom bits, block ciphers transform data in bigger chunks (called blocks).

  In this lecture we'll look at modern block ciphers and how they are designed.
- Security Text 2025.01.05.txt, around line 16:
  In this lecture we'll look at modern block ciphers and how they are designed.

  We'll also introduce the key security property we need for encryption (left informal before) which is semantic security.

  Finally we'll talk about message integrity and the need for message authentication codes (MACs) and how to combine them with encryption to get authenticated encryption (AE) which is the construction we almost always want in practice.
- Security Text 2025.01.05.txt, around line 39:
  Block ciphers and pseudorandom permutations

  The transition from a monoalphabetic cipher, which operates on characters individually, to a bigram-based cipher like the Playfair cipher suggests the power of operating on larger blocks of plaintext at once.

  The modern version of this is the block cipher. Playfair is a crude block cipher where blocks are 2 letters each, modern block ciphers typically operate on blocks of 128 or 256 bits of data. Block ciphers define a permutation which maps blocks of plaintext to blocks of ciphertext. Specifically, block ciphers are defined as a function:
- Security Text 2025.01.05.txt, around line 76:
  C2

  This is called electronic code book (ECB) mode and is a natural first attempt at encrypting data. Unfortunately, it does not provide very good confidentiality: any two plaintext blocks which are identical will produce an identical ciphertext block, so the adversary can tell which blocks of plaintext are identical. This is the exact same weakness of classic monoalphabetic cipher, only a cryptanalyst can perform frequency analysis of blocks rather than letters.

  Depending on what type of data one is encrypting, this can be a severe problem. Observe the following image (of Tux the Linux penguin) encrypted under ECB mode and a randomized stream cipher:
- Security Text 2025.01.05.txt, around line 78:
  This is called electronic code book (ECB) mode and is a natural first attempt at encrypting data. Unfortunately, it does not provide very good confidentiality: any two plaintext blocks which are identical will produce an identical ciphertext block, so the adversary can tell which blocks of plaintext are identical. This is the exact same weakness of classic monoalphabetic cipher, only a cryptanalyst can perform frequency analysis of blocks rather than letters.

  Depending on what type of data one is encrypting, this can be a severe problem. Observe the following image (of Tux the Linux penguin) encrypted under ECB mode and a randomized stream cipher:

- Security Text 2025.01.05.txt, around line 81:
  Semantic security: the gold standard for encryption Clearly the security of a block cipher in ECB mode leaves something to be desired. But we haven't yet specified a concrete security goal for encryption schemes. The standard security property is semantic security. It is defined by the following game:

6. CBC-MAC

Alice computes a MAC by running CBC encryption and keeping only the final block (tag). Let the block cipher be E_K, IV be public, and messages be multiples of the block size: for a message with blocks $(m_0, \ldots, m_{\ell-1})$:

$$c_{-1} = IV, \quad c_i = E_K(m_i \oplus c_{i-1}), \quad \text{Tag}(M) = c_{\ell-1}.$$

This scheme is insecure if the IV is not fixed to zero and/or message lengths are not fixed. We show existential forgeries using only observed tags.

Forgery from two known (IV, Tag) pairs

Suppose we know (IV_1, T_1) for message $M_1 = (m_0^{(1)}, \ldots, m_{a-1}^{(1)})$ and (IV_2, T_2) for message $M_2 = (m_0^{(2)}, \ldots, m_{b-1}^{(2)})$. Consider the crafted message

$$M_3 = M_1 \parallel (m_0^{(2)} \oplus IV_2 \oplus T_1) \parallel m_1^{(2)} \parallel \cdots \parallel m_{b-1}^{(2)}.$$

Then CBC chaining over M_3 with IV$= IV_1$ yields $c_{a-1} = T_1$ and the next block input equals $m_0^{(2)} \oplus IV_2$, so the remainder of the computation exactly reproduces the path that produced T_2. Hence

$$\text{Tag}(M_3) = T_2, \qquad \text{using } (IV_1, T_2).$$

This is a valid existential forgery unless M_3 was previously MACed.

One-block forgery (when a single-block tag is known)

If an observed message M has a single block x with pair (IV, T), then $T = E_K(x \oplus IV)$. For any chosen block y, define

$$M' = x \parallel (y \oplus IV \oplus T).$$

Its tag under IV is $E_K(y \oplus IV)$, i.e., the tag that would be produced for the one-block message y. Thus we can forge a valid tag for y without the key.

Why this fails

CBC-MAC is only secure when the IV is fixed (often zero) and the message length is fixed and included in the computation; otherwise, the chaining malleability above enables splice-and-extend forgeries.

Citations to provided Notes

- Notes 2025.01.05: CBC mode definition, pitfalls, and MAC constructions; AE with separate keys.
- Notes 2025.01.03/01.05: MAC goals (EUF-CMA) and why naive combinations fail.