4/6/2024

CMSC414 Midterm 2 Review

DISCLAIMER: THIS MATERIAL IS MADE BY A STUDENT AND MAY CONTAIN ERRORS. CONSUME WITH A “GRAIN OF SALT” AND PLEASE CONSULT DAVE OR A TA WITH ANY QUESTIONS

RISHABH BARAL

AUTHOR'S NOTE TO READERS

Dear Students,

As we approach the midterm examination for CMSC414: Principles of Network Security, I wanted to provide some additional guidance to aid in your preparation. This review guide is intended to assist you in solidifying your understanding of the key topics that will be covered in the upcoming assessment.

Symmetric Key Cryptography serves as the cornerstone of secure communication in network environments. Understanding its principles, algorithms, and applications is crucial for implementing robust encryption mechanisms. Be sure to review concepts such as encryption, decryption, key management, and cryptographic protocols thoroughly.

Public Key Cryptography and the intricacies of Public Key Infrastructure (PKI) introduce a paradigm shift in cryptographic systems. Delve into the concepts of asymmetric encryption, digital signatures, certificate authorities, and trust models. Mastery of these topics will enable you to appreciate the security mechanisms underlying digital communication infrastructures.

Cryptographic Failures highlight the importance of analyzing vulnerabilities and weaknesses in cryptographic systems. Explore common pitfalls such as weak key generation, algorithmic flaws, and implementation errors. Developing a critical eye towards cryptographic failures will enhance your ability to design resilient security solutions.

Anonymity in network communication presents both challenges and opportunities in preserving privacy and confidentiality. Examine techniques such as anonymizing networks, pseudonymity, and privacy-enhancing technologies. Understanding the principles of anonymity will empower you to navigate the complexities of privacy in networked environments.

I encourage you to engage actively with the course material, collaborate with your peers, and seek clarification on any concepts that may seem ambiguous. Remember that a solid foundation in these fundamental principles will not only serve you well in the upcoming midterm but also lay the groundwork for your continued success in the field of network security.

Best regards,

Rishabh Baral

Contents

Symmetric Key Cryptography

Symmetric Key Cryptography, often referred to as secret key cryptography, is a fundamental concept in the field of information security. It plays a pivotal role in ensuring the confidentiality and integrity of data exchanged over insecure channels. In this exposition, we will delve into the principles, mechanisms, and practical applications of symmetric key cryptography.

At the heart of symmetric key cryptography lies the notion of a shared secret key between communicating parties. This key serves a dual purpose: encryption and decryption. The encryption process involves scrambling plaintext into ciphertext using the shared key, while decryption reverses this process, transforming ciphertext back into plaintext.

The strength of symmetric key cryptography hinges on the secrecy and randomness of the shared key. Therefore, the key must be kept confidential and changed periodically to mitigate the risk of compromise. Additionally, the key space, i.e., the total number of possible keys, should be sufficiently large to thwart brute-force attacks.

Symmetric key cryptography employs various algorithms and techniques to achieve secure communication. One such algorithm is the Advanced Encryption Standard (AES), which is widely adopted due to its robustness and efficiency. AES operates on fixed-size blocks of data and supports key lengths of 128, 192, or 256 bits.

Another notable mechanism is the Data Encryption Standard (DES), although its usage has waned due to its susceptibility to brute-force attacks. DES operates on 64-bit blocks of data and utilizes a 56-bit key. Despite its vulnerabilities, DES laid the groundwork for modern symmetric key algorithms.

Consider Alice and Bob, two parties wishing to communicate securely over an insecure channel. To establish a secure connection, they first agree upon a secret key using a secure channel or a trusted third party. Once the key is established, Alice encrypts her message using the shared key and sends the ciphertext to Bob. Upon receiving the ciphertext, Bob decrypts it using the same key, thereby retrieving the original message. Just as with any secure communication, recall that we are trying to maintain the CIA of Network Security (**C**onfidentiality, **I**ntegrity, and **A**uthenticity). Essentially, regardless of what encryption tactic we use in the creation of the symmetric key, we want to be able to keep others from reading the messages or data communicated (Confidentiality), we want to keep others from *imperceptibly* tampering with the messages or data communicated (Integrity), and we want to keep others from *undetectably* impersonating either of the communicating parties (Authenticity).

In the realm of symmetric key cryptography, several essential "black boxes" play instrumental roles in securing communication and data integrity. These black boxes encompass Block Ciphers, Message Authentication Codes (MAC), Hash Functions, and the Diffie-Hellman Key Exchange. Each serves a distinct purpose, contributing to the overall robustness and efficiency of cryptographic systems.

Block Ciphers are cryptographic algorithms that operate on fixed-size blocks of data, transforming plaintext into ciphertext using a shared secret key. They provide confidentiality and are integral to symmetric encryption schemes such as AES (Advanced Encryption Standard) and DES (Data Encryption Standard). **Message Authentication Codes (MAC)** ensure data integrity and authenticity by generating a tag, or MAC, which is appended to the message. This tag is computed using a secret key and the message itself, allowing recipients to verify the integrity and origin of the received data. HMAC (Hash-based Message Authentication Code) is a popular MAC construction based on cryptographic hash functions. **Hash Functions** are one-way functions that map arbitrary-sized input data to fixed-size output, known as hash values or digests. These functions are employed in various cryptographic applications, including data integrity verification, password hashing, and digital signatures. A robust hash function exhibits properties such as collision resistance and preimage resistance, ensuring the security of cryptographic protocols. **Diffie-Hellman Key Exchange** is a cryptographic protocol used to establish a shared secret key between two parties over an insecure channel. Unlike symmetric key exchange methods, such as pre-shared keys, Diffie-Hellman enables parties to negotiate a shared key without prior communication. This protocol relies on the discrete logarithm problem for its security and forms the basis for many secure communication protocols, including SSL/TLS. Together, these black boxes form the building blocks of symmetric key cryptography, providing the necessary tools to achieve confidentiality, integrity, and authenticity in digital communication. Understanding their principles and functionalities is essential for designing and implementing secure cryptographic systems in practice.

Symmetric key cryptography is a cornerstone of modern information security, facilitating secure communication and data protection. By leveraging shared secret keys and robust encryption algorithms, it enables parties to exchange sensitive information with confidence. However, its efficacy depends on proper key management practices and the judicious selection of encryption algorithms. As adversaries continue to evolve, so must our cryptographic techniques to ensure the continued security of digital communications.

Public Key Cryptography

However easy and quick symmetric-key cryptography may be it has three main shortcomings. Firstly, it requires pairwise key exchanges, which can run as high as O(n^2) if the network being studied is an all-to-all network such as a chatroom or email service. Secondly, the use of symmetric-key cryptography requires that both users be online in order to preserve authenticity. An example is if one user uploads a document to a popular document hosting site and then goes offline *forever*, no other user (apart from the original uploader) can be certain that the document was really uploaded by the uploader. Lastly, using black-box methods such as Diffie-Hellman prevents eavesdropping, preserving confidentiality. However, these methods are not resilient to tampering meaning that integrity is not preserved.

Public Key encryption comprises three separate algorithms. Firstly, the key generation algorithm used is often a *randomized* algorithm with a **nondeterministic** output making it difficult to infer the Secret Key (key known by only one person) from the Public Key (the key shared with all users on a network). The thing that makes public key encryption slightly stronger than symmetric key encryption is that the public and secret keys are bound together in such a way that for every public key, there is **a single** secret key. Secondly, the message is encrypted using the **public key** of the intended recipient. This step takes a message of a certain length, encrypts it using the public key of the intended recipient in some way, and then returns a ciphertext that is **the same length** as the original message. The last step in public key encryption is the decryption itself. The decryption depends on the recipient's **secret key** which, in an ideal situation, should only be known by the intended recipient. This algorithm, unlike the previous two steps, **IS** deterministic, meaning that any ciphertext, if decrypted correctly, should return the original message. Furthermore, to ensure security when using public-key cryptography, the encryption method should appear almost random meaning that small changes in the plaintext should, theoretically, result in large changes in the ciphertext. All things considered, the encryption function used should approximate somewhat of a one-way trapdoor (a function with no backdoor that cannot be decrypted without the secret key used).

An alternative to both public-key cryptography (it can often be slow) and symmetric-key cryptography (it may not be very secure) is the use of what is known as Hybrid Encryption. Hybrid encryption follows almost the same process as public-key encryption, except that it adds an extra step in which a symmetric key is generated and used for the decryption as opposed to a secret key. In this way, it combines the best attributes of both methods by preserving the security and authenticity of the public-key encryption and combining it with the speed and randomness of the symmetric-key encryption.

Just as with all encryption, the goals of public-key encryption are not only to preserve the content and integrity of the message, but also to be able to define with certainty the sender of the message. In public key encryption, the authenticity of the messages sent on a network are determined by a set of digital signatures which use the SECRET KEY, meaning that only the sender can ever verify a message they sent.

Public Key Infrastructure

The main question to answer here is “How does a user know **for sure** who they are communicating with?” Let’s look at an example that will provide some insight into the matter. Let's say you’re on your favorite browser, Google Chrome, and that it’s your birthday. You want to see how much “Birthday Money” your relatives have sent you, so you log in to your bank, Bank of America. Now how do you know that you’re communicating with Bank of America itself and not a mockup? This is where a digital signature comes in. Bank of America has a Public Key that it sends to a Certificate Authority (let’s assume it’s VeriSign in this case). The Certificate Authority will then attempt to verify the authenticity of the public key and (let’s assume it’s a correct key) issue a certificate. Behind the scenes, your browser (Chrome) receives the key and certificate from Bank of America verifying that the website is legitimate. This is an instance of what is known as the Public Key Infrastructure or PKI. Now Bank of America is verified as legitimate by the certificate from VeriSign. However, can you trust VeriSign? This is the second step of the PKI in which the Certificate Authority (VeriSign in this case) is certified by another Certificate Authority (let's use Symantec as the second CA). Now how do you know whether Symantec is legitimate? This is the third and final step of the PKI, known as a root certificate, in which the certificate authority provides **self-verification** which is often irrefutable.

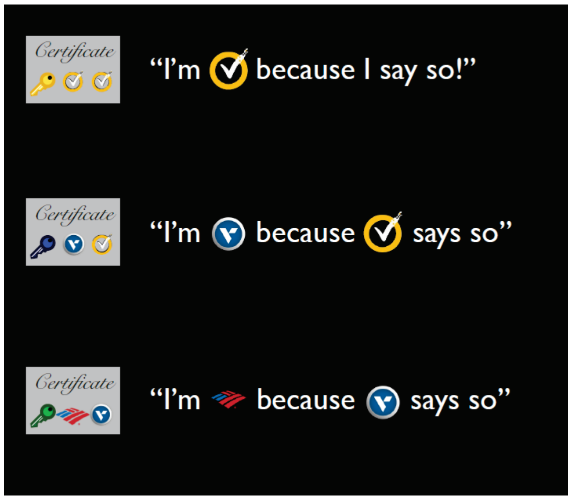


Figure 1: The PKI and Verification Process (Symantec is what is called the Root Certificate)

However, there are many root certificates and tracking them all can be difficult or even impossible. For this reason, all modern devices have a **single** root key store, which stores all root certificates. **THIS IS VITAL TO SECURITY, SO IT CANNOT CONTAIN ANY MALICIOUS CERTIFICATES.**

The possibility of malicious certificates raises an interesting question: “What happens if a certificate becomes invalid for any reason?” This is the beginning of a process known as certificate revocation in which the website asks the CA to revoke its certificate with an intent to reissue an updated certificate in the future. This is vital to security and in order to be as safe as possible, websites should request revocations as quickly as possible whilst browsers should check for revoked certificates constantly, obtaining revocations as soon as possible. This fact becomes more prevalent when considering an attack like Heartbleed. When the “Heartbleed” issue was discovered, every vulnerable website should theoretically have patched, revoked, and reissued their certificates. In this way, Heartbleed sort of became an industry experiment on how quickly and thoroughly administrators react to issues. With regards to certificates, events were described as “Heartbleed induced” if the certificate was reissued on or after April 7th, had a validity of more than 60 days, and the domain reissued the certificate less than once every two months. What this testing showed was generally positive as only 7% of domains in the Alexa dataset were still vulnerable 3 weeks removed from the patching. With regards to revocations, the statistics were somewhat concerning with 13% of websites revoking their certificates and 27% reissuing them.

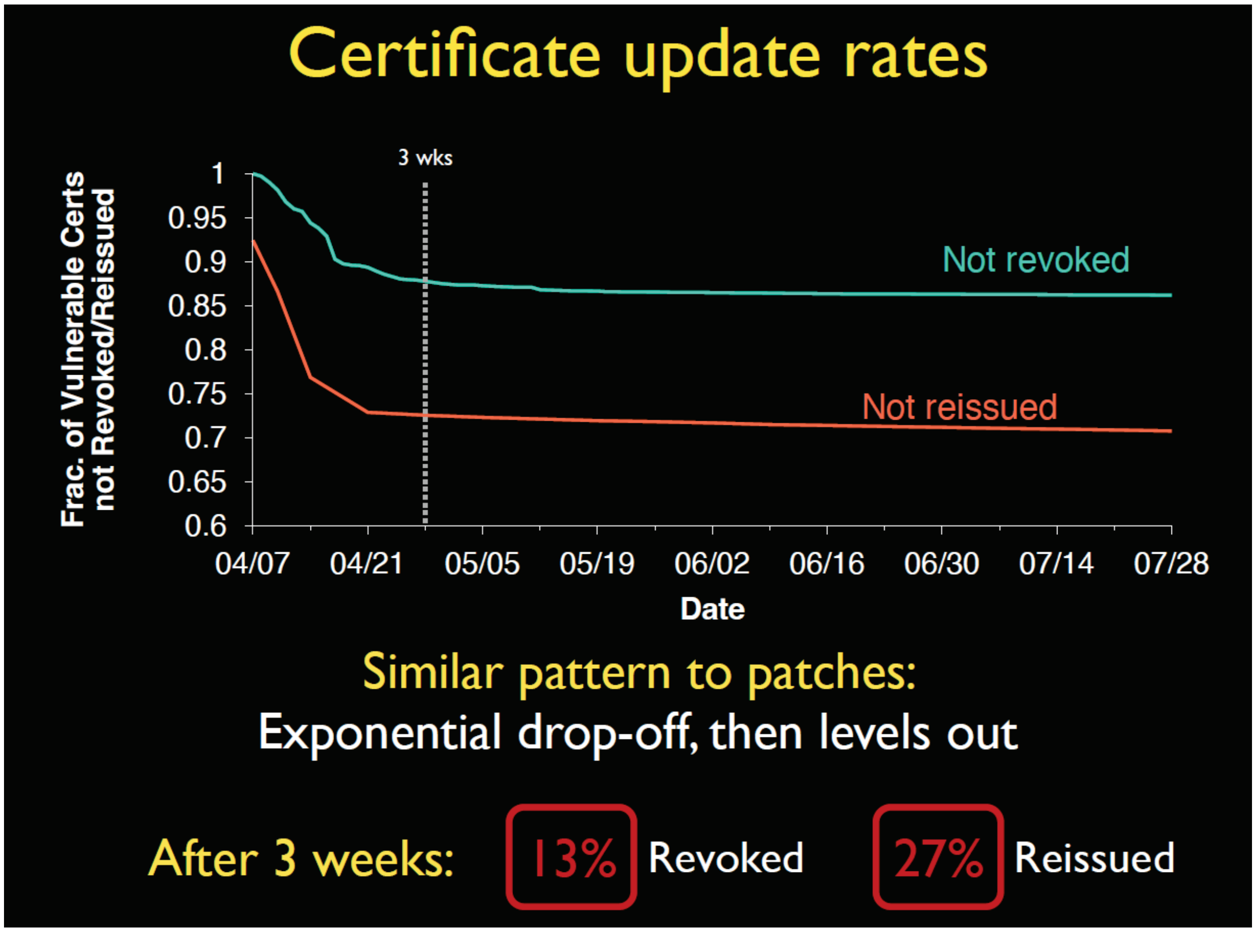


Figure 2: Graph showing the rates of Revocation and Reissue of certificates after identification of Heartbleed

Another somewhat concerning statistic is that of the certificates issued by various websites, 60% of retired certificates were never revoked. Currently, approximately 8% of vulnerable certificates still haven't expired, meaning that Heartbleed could be a continued problem for years to come.

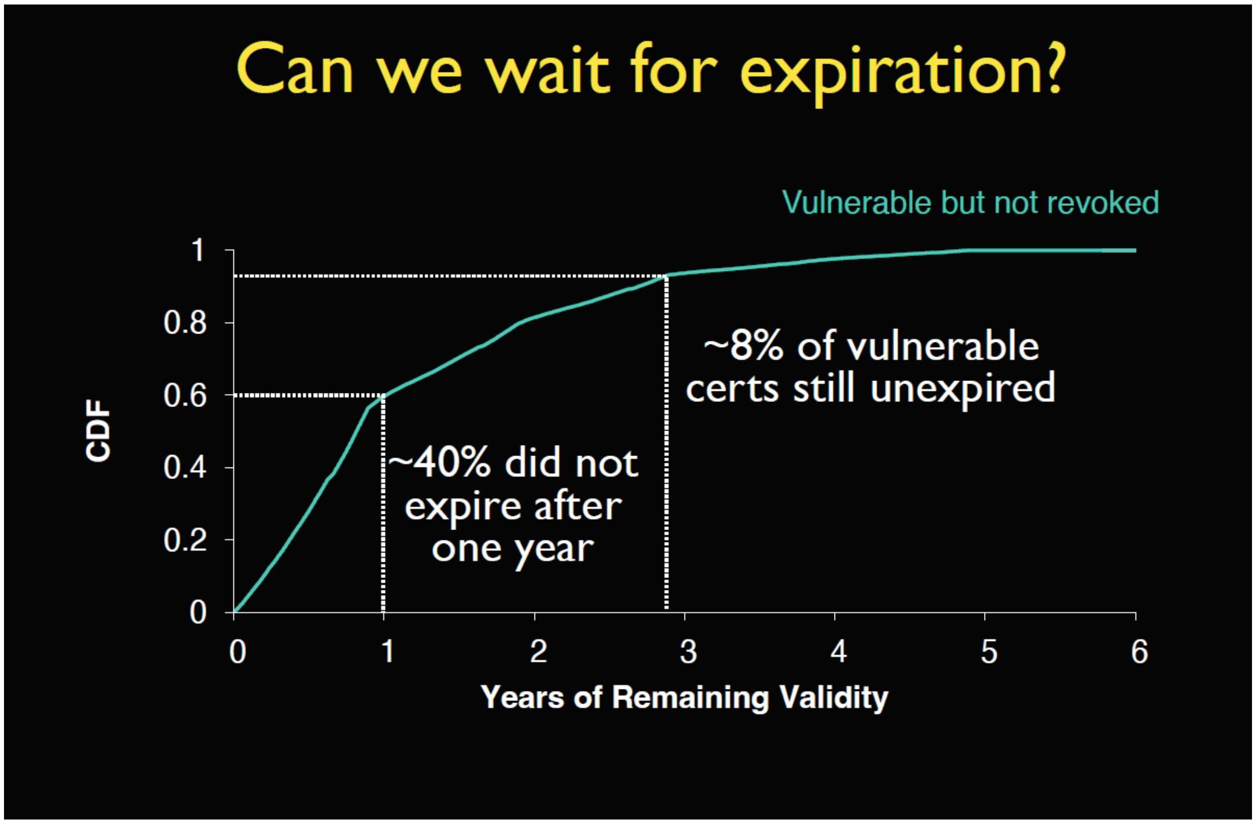


Figure 3: Graph showing the rate of Revocation for vulnerable certificates based on Expiration

The other concern with today's PKI (which we are all often visiting without even realizing) is the use of a CDN (**C**ontent **D**elivery **N**etwork), a third-party hosting service that has varying levels of involvement but is trusted for one single purpose: **delivering fast and accurate content.** The security concern here is that third-party CDNs know the private key of each of their customers, creating a security nightmare in the event that a CDN is hacked. The only way to verify the authenticity of a CDN is through what is known as a Subject Alternate Name (SAN) list. In spirit, this list is supposed to have *multiple names* for the ***same organization.*** However, the SAN list is often seen as multiple **unrelated** domains lumped together (since they use the same CDN).

How Security Fails in Practice

There is a well-known saying pertaining to security. It goes something like this: "A chain is only as strong as its weakest link." Nowhere is this idea encapsulated more than in the failures of security in different real-world scenarios. For example, there are currently 15,134 apps on the Google Play Store that use some form of cryptography. Of these 15000+ apps, almost 11,800 were analyzed. In that analysis, it was revealed that nearly 5700 (about half) used a mode of encryption known as ECB. Now, we've all seen the famous image of Tux the penguin (the lovable linux mascot) encrypted using ECB. If you look at it closely, you can see that it isn't very encrypted at all. You can still tell that it is an image of the beloved Tux. This isn't so much because ECB is bad (it is), but rather because ECB used a strong cryptographic technique (block ciphers) in a poor way.

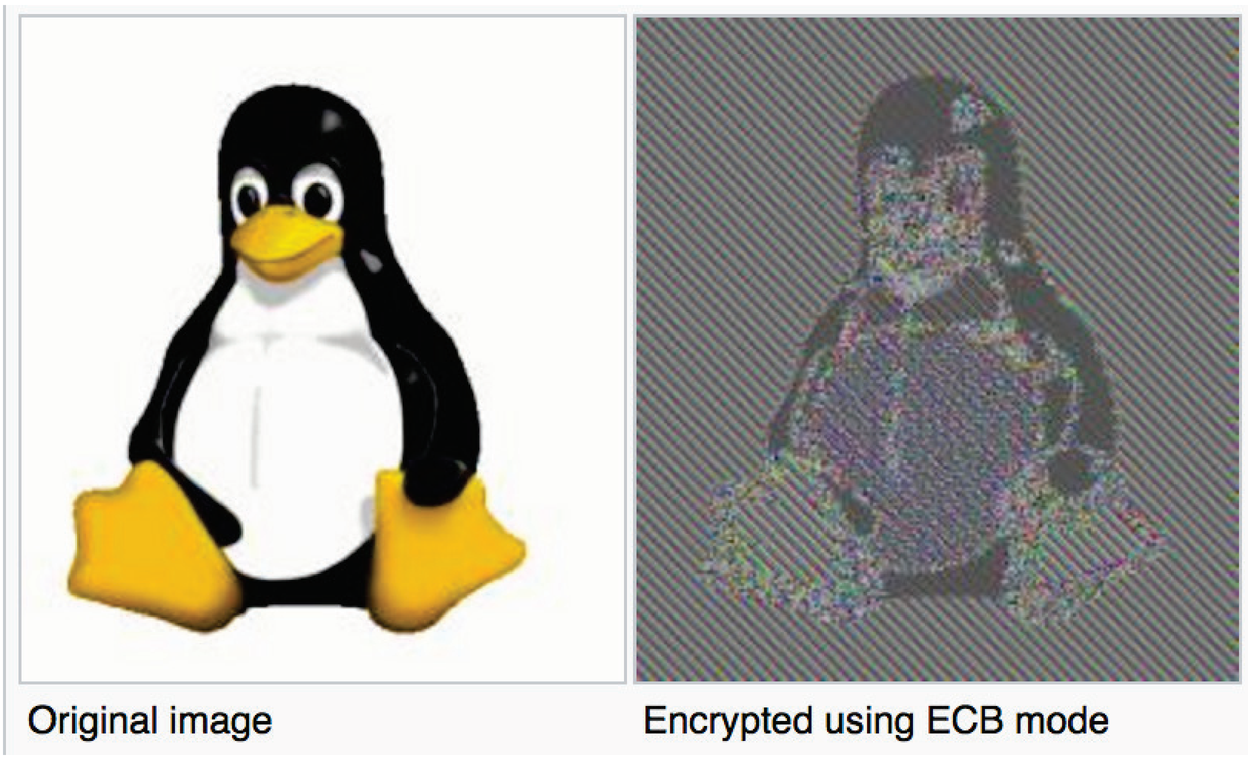


Figure 4: Tux the Linux Penguin Encrypted using ECB

Another way in which poor encryption can create a security headache is the use of a CDN. Often times, when using a CDN multiple companies using that CDN will all share a key. This makes CDNs a prime target for encryption-based attacks since they are all based on one key, meaning attacking a CDN can attack multiple companies, creating mass havoc.

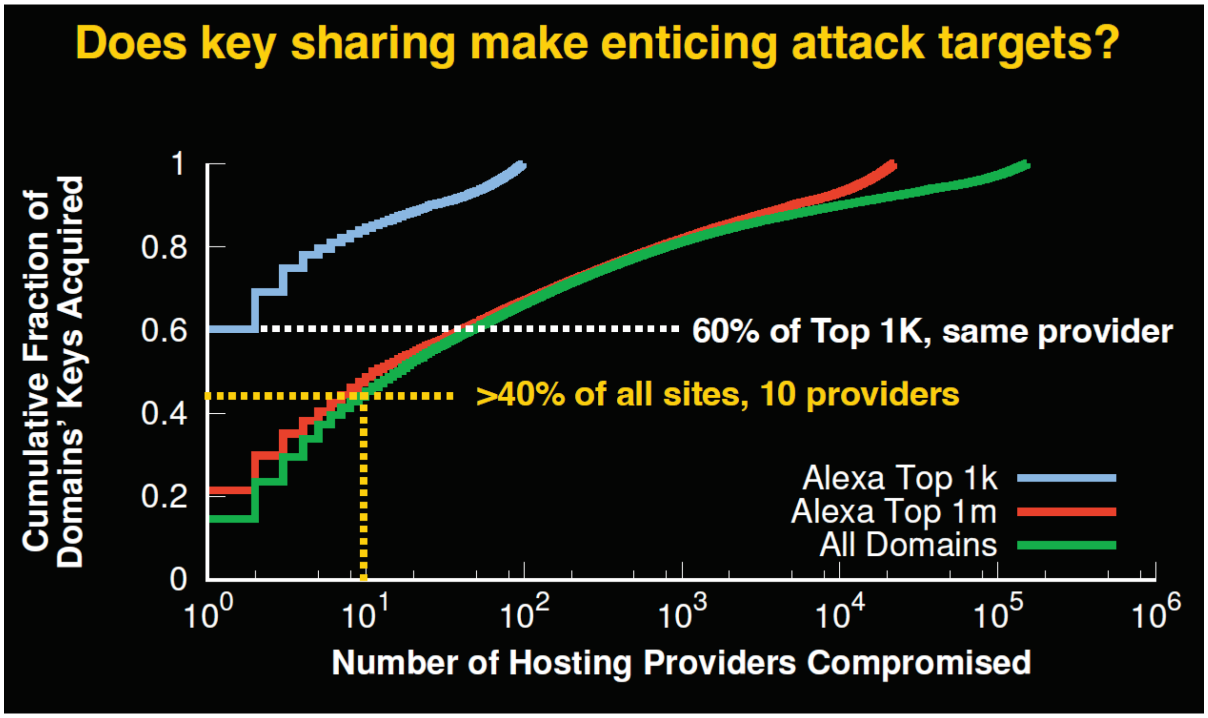


Figure 5: Graph showing how attacking a CDN can cause havoc on the internet

Anonymity

Let's imagine a scenario. You're James Bond, the famous 007. However, you're lost deep undercover on a mission, and you need help from headquarters. Because you're undercover, you don't want your identity to be revealed. Also at HQ, you want your request to be classified, so nobody knows what trouble you are in. This is the main purpose of anonymity in security. The first situation (you don't want to reveal your identity) is called sender anonymity whereby there could be a large number of senders and any possible attacker cannot reliably infer who the sender was. The second scenario (you don't mind revealing your identity in lieu of your troubles being private to your team) is called receiver anonymity whereby the sender is known but the receiver is unknown.

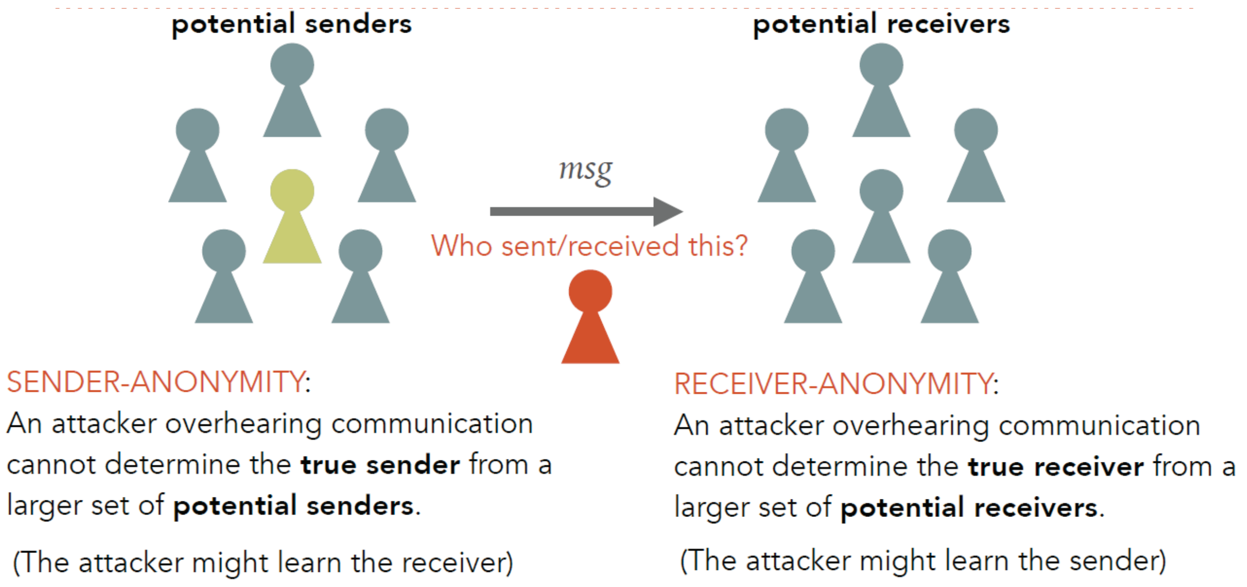

potential senders
potential receivers
msg
Who sent/received this?
SENDER-ANONYMITY:
An attacker overhearing communication cannot determine the **true sender** from a larger set of **potential senders**.
(The attacker might learn the receiver)
RECEIVER-ANONYMITY:
An attacker overhearing communication cannot determine the **true receiver** from a larger set of **potential receivers**.
(The attacker might learn the sender)

Figure 6: A More Thorough Explanation of Sender and Receiver Anonymity

As a last consideration of security, let's consider what are known as mix-nets (short for **Mix**er **Net**work**s**) In such a situation, let's assume that an attacker can eavesdrop on every possible link in the mix-net.

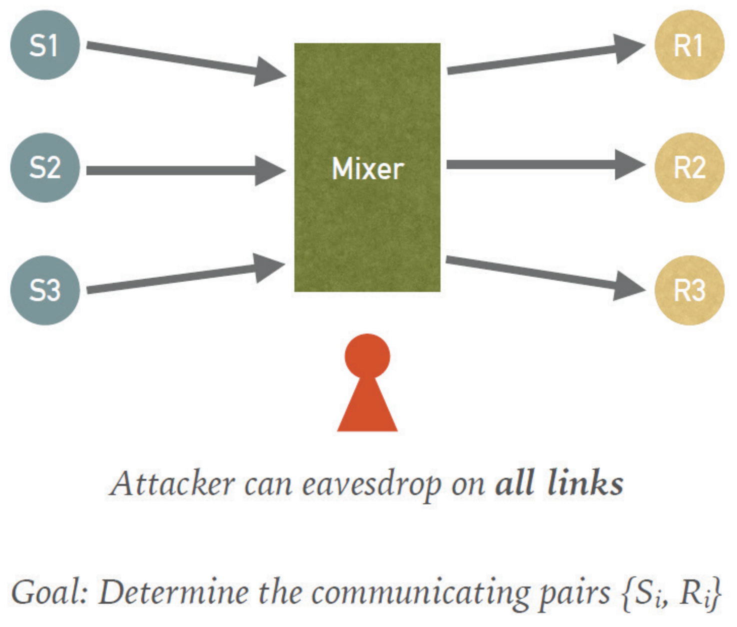

S1
S2
S3
Mixer
R1
R2
R3
Attacker can eavesdrop on **all links**
Goal: Determine the communicating pairs $\{S_i, R_j\}$

Figure 7: Simple Rendition of a Mix-Net (3 senders and 3 receivers)

In this case, sending all messages unencrypted would be catastrophic since there would be **absolutely no security**.

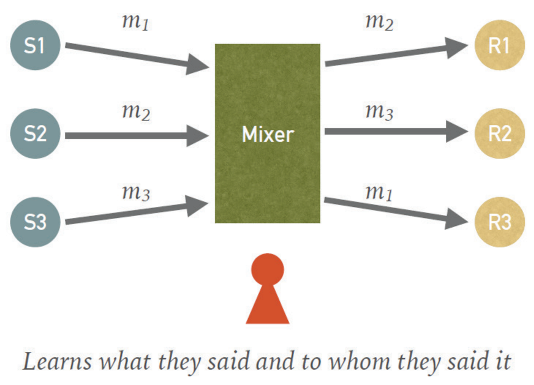


Figure 8: Diagram of the catastrophe of sending unencrypted messages on a compromised mix-net

Encryption may not be the best practice, but it can help if used correctly.

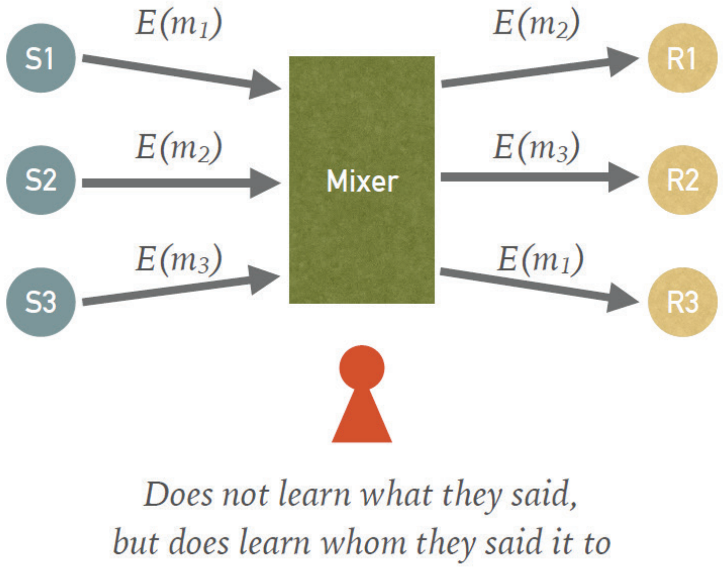

S1
S2
S3
$E(m_1)$
$E(m_2)$
$E(m_3)$
Mixer
$E(m_2)$
$E(m_3)$
$E(m_1)$
R1
R2
R3
Does not learn what they said, but does learn whom they said it to

Figure 9: An example of encryption used POORLY in a mix-net

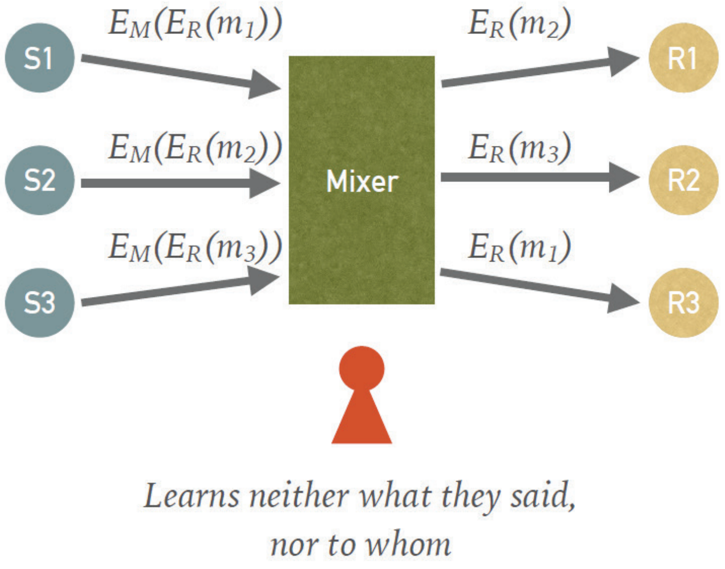
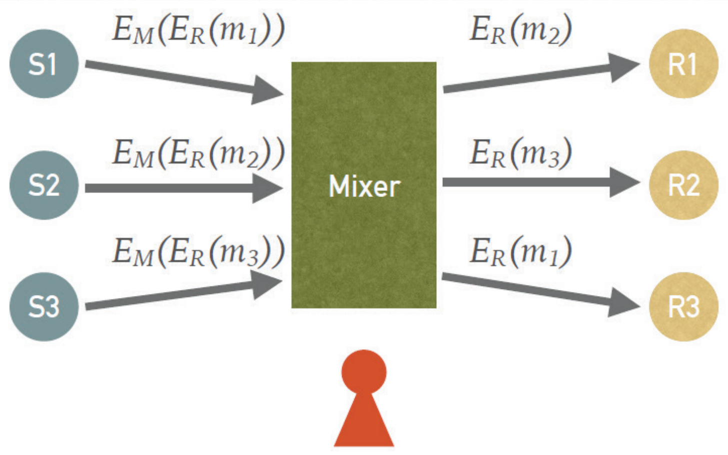


Figure 10: An example of encryption used WELL in a mix-net



What if the messages arrive at different times?

***Mix** the order of delivery*

Figure 11: An example of encryption used PERFECTLY in a mix-net (Delayed Delivery for messages arriving at different times)