How Internet Encryption Works and Why It's important

When you tell someone a secret, you trust them to keep it. If that person were to tell someone your secret, they would lose your trust. However, what if someone overheard you telling that person your secret without you even knowing?

Daily, we trust our computers and the internet to keep our personal information safe. As more transactions become electronic, more sensitive information traverses the internet and fills databases. That is why cyberattacks are so prevalent. With valuable information at their fingertips, hackers can sell your information on the dark web, leak classified documents, demand ransom for information or photos, and track movements and activities. Hackers will use linked payment accounts to shop, expose your intellectual property, and steal your identity. Luckily, there is a way to prevent these nefarious activities.

Are you wondering about the lock symbol in your browser search bar? With the advent of modern computers, the need to prevent data theft is critical. In 1975, the U.S. published the first encryption standard to protect private data. A hacker successfully cracked the code in 1997, generating the need for a new encryption standard. The cryptographic protocol used then was Secure Sockets Layer (SSL). It was superseded by Transport Layer Security (TLS) in 1999. There are several, stark differences between the two protocols, but both aim to achieve the same goal — secure data transport on the Internet. Although SSL is outdated and

KEYSIGHT TECHNOLOGIES

insecure compared to TLS, it is still common to see SSL and TLS used interchangeably when discussing internet encryption.

Fast forward to 2018 – the release of the most recent internet encryption standard. The Transport Layer Security Standard version 1.3 evokes a new era of encryption that makes the internet a safer place to exchange data. This white paper will discuss the history of cryptography, how internet encryption works, the new requirements in TLS 1.3, passive versus active SSL, and how to successfully deploy a security infrastructure that can handle and inspect encrypted traffic.

A Brief History of Cryptography

Cryptography is traceable back to the ancient Egyptians who carved non-standard hieroglyphs into the walls of tombs. This is the first known example of random symbols or numbers used to conceal the meaning of a message. Eighteen hundred years later, Ancient Romans were the first on record to use a substitution cipher. Julius Caesar reportedly used the Caesar cipher algorithm to encrypt messages for his military generals should the enemy intercept their correspondence. The algorithm involves shifting the letters of the original message a certain number of steps. That way the message appears as a meaningless jumble of letters. The number of steps to shift, also referred to as the key, is only known and agreed upon by the sender and receiver of the message. Figure 1 depicts this technique and how it encrypts the message "glory," using a key of 4.

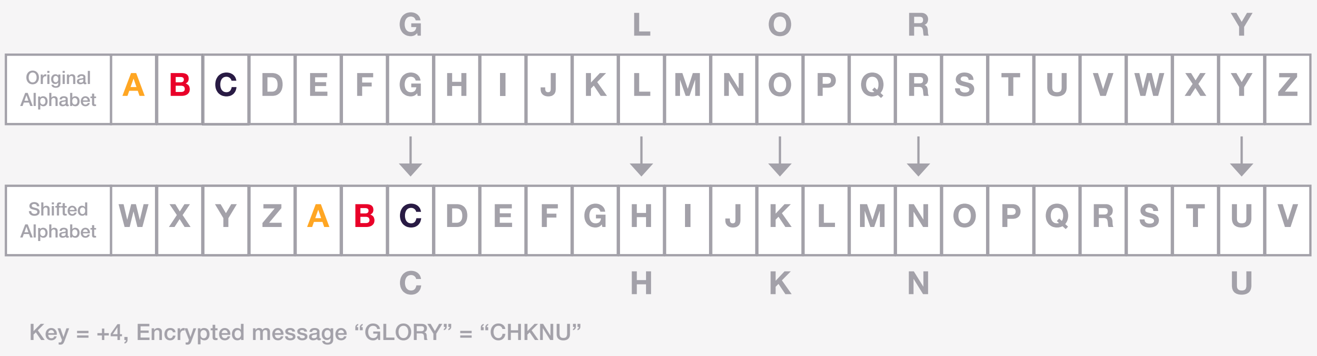
G L O R Y
Original Alphabet
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Shifted Alphabet
W X Y Z A B C D E F G H I J K L M N O P Q R S T U V
C H K N U
Key = +4, Encrypted message "GLORY" = "CHKNU"

Figure 1. Caesar cipher substitution cipher example

As clever as this substitution cipher is, it isn't hacker-proof. Simply shifting the alphabet a maximum of 26 times will eventually reveal the hidden message. It could easily take under an hour to solve. Although this method of encryption is not the most secure, it is the basis for the way we encrypt our internet today.

Encryption Keys Today

Today, encryption keys are a little harder to crack than the previous Caesar cipher example. They are composed of a string of ones and zeroes that usually range anywhere from 128 to 4,096 bits. Encryption keys have an astronomical number of key combination possibilities. For example, a 256-bit key made of ones and zeroes will have 2^{256} or 1.2×10^{77} key possibilities.

Let's put this into perspective:

As of 2018, the fastest supercomputer on Earth, the IBM Summit, performs 2×10^{17} calculations per second.[1] That means it takes the Summit approximately 1.9×10^{52} years to run through every possible key combination. Our universe is only 1.38×10^{10} years old.[2] An encrypted message is sent and read in a matter of seconds. So, even cracking the key in a fourth of that time would reveal a very, very old message.

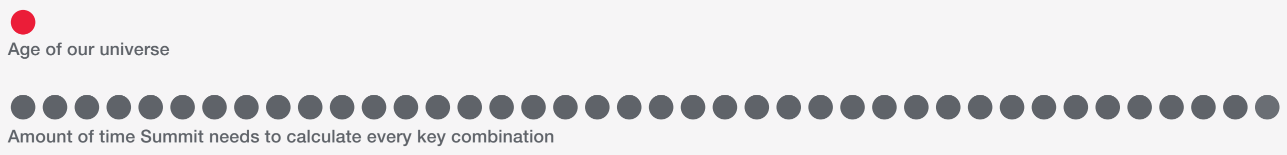

Figure 2. A visual representation of the age of our universe versus the time it takes the world's fastest supercomputer to compute every possible key combination for a 256-bit key

Computer technology and capabilities are growing exponentially, and soon, 256-bit keys will not be strong enough. The National Institute of Standards and Technology predicts that 2,048-bit keys will be decipherable by the year 2030 and recommends using 3,072-bit keys for encryption after 2030.[3] Luckily, our method of encrypting the internet involves deriving more than one key and makes hacking an encrypted session that much harder.

1 John Diente, "US Has World's Fastest Supercomputer Again: IBM'S Summit Can Do 200 Quadrillion Calculations Per Second," Tech Times, June 9, 2018, https://www.techtimes.com/articles/229816/20180609/us-has-worlds-fastest-supercomputer-again-ibms-summit-can-do-200-quadrillion-calculations-per-second.htm

2 Nola Taylor Redd, "How Old is the Universe?" Space.com, June 8, 2017, https://www.space.com/24054-how-old-is-the-universe.html

3 Elaine Barker, Department of Commerce, National Institute of Standards and Technology, Recommendation for Key Management Part 1: General, Special Publication 800-57 Part 1 Revision 4, January 2016, pg. 53-58, https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

Symmetric and Asymmetric Keys

When both the sender and receiver agree ahead of time on a key for a message, as in Caesar's cipher, it is called symmetric encryption. Unfortunately, using this type of premeditated key makes it impossible to communicate between sender and receiver in today's open internet, unless the two parties meet in person. Asymmetric encryption resolves this issue.

Asymmetric encryption involves two keys — one for encryption (public) and one for decryption (private). The public key is open to the public which anyone can use to encrypt a message. The private key is for decryption and is only used by those who know it. Asymmetric encryption allows messages to move back and forth in a session without having to agree on a private key. Sounds odd, doesn't it?

Finger painting easily explains the process. The logic behind asymmetric encryption is based on a one-way function — a function that is easy to solve in one direction but hard to solve in the reverse direction. Take mixing colors, for example. It is easy to mix light blue and dark red to make purple but hard to unmix purple into the exact original two colors.

Here is an example:

Let's say Alice and Bob want to share a secret color that they don't want anyone else to see. First, they each agree to a starting color that anyone can publicly see, say yellow. Second, Alice and Bob randomly select each of their own private colors to mix with yellow. Alice chooses red, and Bob wants blue. Alice's mixture turns orange, and Bob's turns green. Both mixtures disguise each of their private colors. Third, Alice sends her orange mixture to Bob, and Bob sends his green mixture to Alice. Someone from the outside looking at this exchange sees the colors yellow, green, and orange, but they cannot see the private colors.

Finally, the magical step of the exchange: both Alice and Bob add their private colors to the mixture they received. Alice adds red to the green mixture, and Bob adds blue to the orange mixture. The final mixtures reveal the same brown-hued color for both Alice and Bob, their shared secret color. That person watching from the outside cannot see the shared secret color because they do not know what colors Alice and Bob added in private. Figure 3 depicts this ingenious exchange.

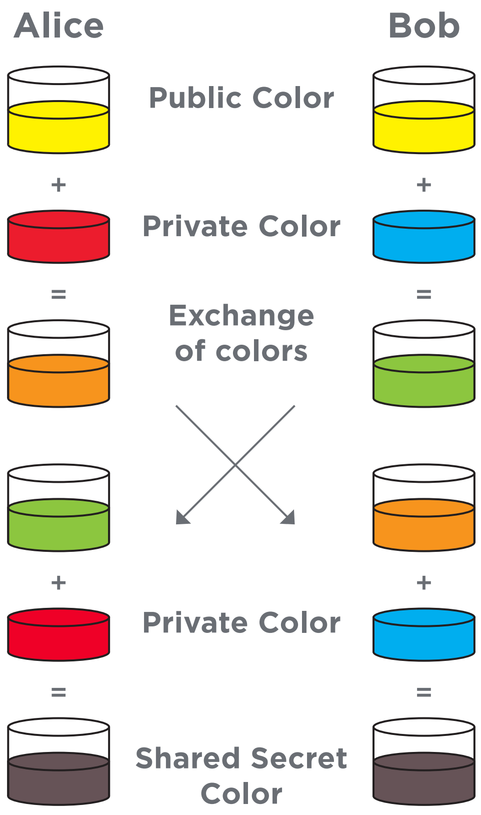

Figure 3. Illustration of the secret color exchange between Alice and Bob

The mixed color exchange also provides a way to authenticate the two participating parties. Let's say that after this initial exchange between Alice and Bob, an outsider poses as Bob and sends a green-colored mix to Alice. When Alice adds her private color to the mixture, she does not get the same shared secret color she got before (brown). Because of this, Alice knows this green mixture came from an imposter attempting to hijack the session. So, Alice terminates the communication session.

Now, let's tie this into encryption keys. Asymmetric keys have two parts — an exponent and a modulus, written as "(exponent, modulus)". So, two keys contain four parts in total. Two main algorithms that take advantage of the one-way function logic generate these keys. They are known as the Rivest, Shamir, and Adelman (RSA) algorithm and the Diffie-Hellman (DH) algorithm. Both use modular arithmetic for key generation but deploy the keys differently for encryption. If you are not familiar with modular arithmetic, Khan Academy has a great overview video. First, we will discuss the RSA method, then DH.

RSA Key Generation

Before TLS 1.3, the RSA algorithm was the widely accepted method for key generation. With RSA, public and private keys generate only once at the initial connection attempt between a sender and a receiver. Then each party stores the keys for later communication. These are known as static keys. In most cases, the sender is a client trying to access a web page, and the receiver is the server that hosts the web page. Both the client and server have one public and one private key each. So, in total, there are four different keys composed of eight parts. Let's walk through the calculations for RSA key generation.

Server public key modulus formation. Deriving the modulus (m) for the public key involves large, prime numbers picked randomly. For the sake of simplicity in this paper, we will pick small prime numbers, 5 and 11. These numbers equal the modulus 55 when multiplied together. The modulus is the same for the private key. Note that the product of two prime integers is not prime. Therefore, the modulus in RSA key generation is never prime.

Server public key exponent formation. To calculate the public key exponent (e), subtract one from each of the numbers chosen before and multiply those together.

$$F(n) = (5 - 1) \times (11\text{-}1) = 40$$

Now, choose a number that is relatively prime to and smaller than 40. Relatively prime means that the number cannot be divided evenly into 40; like 7. The public key exponent is 7.

The server public key (e,m) is $(7, 55)$

Server private key exponent formation. Calculate the private key exponent (x) by using the following equation:

$$x = \frac{1}{e} modF(n)$$

$$x = \frac{1}{7} mod40$$

Multiply each side by $7mod40$

$$7x\ mod40 = 1$$

Assume $7x = y$

$$y * mod40 = 1$$

$$y = 161$$

$$161 = 7x$$

$$x = 23$$

Since the modulus is the same as the public key, the private server key (x,m) is (23, 55).

In this example, we will only solve for the public and private keys of the server. When the client wants to connect to the server's web page, it first sends the server a message encrypted with the server's public key. This initial message is plaintext since it is not encrypted yet. Once the server decrypts the message, it's known as cleartext, since the message was transmitted, unencrypted, and stored.

Let's say the client wants to send the message "2" to the server. It would use the following equation:

$$\textit{encrypted message} = (\textit{plaintext message})^{e} \textit{ mod } m$$

$$\textit{encrypted message} = 2^{7} \textit{ mod } 55$$

$$\textit{encrypted message} = 128 \textit{ mod } 55$$

$$\textit{encrypted message} = 18$$

Now the server decrypts that message using its private key:

$$\textit{cleartext message} = (\text{encrypted message})^{x} \textit{ mod } m$$

$$\textit{cleartext message} = (18)^{23} \textit{ mod } 55$$

$$\textit{cleartext message} = 743477136140219279133187768832 \textit{ mod } 55 = 2$$

The message was sent and received all without sharing private keys. However, what is stopping someone from calculating those steps backward? The strength of the cryptography lies in the product of the two large prime numbers chosen for the modulus at the start. Like the one-way function color example, these numbers are easily multiplied together to get a product. Someone intercepting the message and looking only at the product would have a hard time figuring out the initial numbers.

You can store RSA encryption keys after their initial inception for use in later communications. Because of this, the RSA encryption method is easy to implement, but its strength is limited. Should someone with bad intentions get ahold of the server's private key, they would be able to decrypt any message ever exchanged between the same parties — from the past or in the future. This flaw in RSA key generation is a significant weakness and prompted a new internet standard. Enter TLS 1.3 and DH.

DH Key Generation & Ephemeral Keys

DH key exchanges are very similar to RSA key exchanges in that they both use modular arithmetic. However, the strength of RSA relies on the modulus computation using the product of two large, prime integers. For DH, on the other hand, the strength lies in its use of discrete logarithms. Let's walk through the math to understand better.

First, the server and client agree publicly on a generator, g, and a prime modulus, m. Note that this modulus is prime, unlike RSA. Let's choose 3 and 23, or 3 mod 23. Second, the server determines a random private number, let's say 7 and computes:

$$3^7 \; mod \; 23 = 2$$

The server then sends this result publicly to the client. Third, the client selects a private number and does the same calculation. Let's say the client chooses 9.

$$3^9 \; mod \; 23 = 18$$

The client sends this result to the server. Someone looking in on this communication session only sees the numbers 2, 18, and 3 mod 23. They cannot see the random private numbers. Lastly, the server and the client each take the number sent to them and raise it to their own private numbers, resulting in the same shared secret number.

$$\text{Server: } 18^7 \; mod \; 23 = 6$$

$$\text{Client: } 2^9 \; mod \; 23 = 6$$

Thus, a messaged is communicated without ever sharing a private key. It may not look like it right away, but both the server and the client did the same calculation with the exponents in a different order. This does not change the final result.

$$\text{Server: } (3^9)^7 \; mod \; 23 = 6$$

$$\text{Client: } (3^7)^9 \; mod \; 23 = 6$$

Someone looking in on this communication cannot discover the secret key without one of the private keys. The strength of this algorithm depends on the quantity of other viable options that can also satisfy the modular arithmetic. This is the discrete logarithm problem. For example, take 3^7 mod 23 = 2. The person snooping in on this conversation can only see 3 mod 23 and 2 since 7 is a private key. Several different numbers could satisfy 3^x mod 23 = 2 like 18, 29, 40, 51 —and the list goes on. This makes computing the exponent and cracking the key challenging for a hacker.

Keep in mind that keys these days come in huge bit sizes so cracking keys using both RSA and DH is challenging.

Originally, these keys were stored for later use, like RSA, which posed as a vulnerability should a hacker obtain a key. This prompted the creation of Diffie-Hellman Ephemeral (DHE). In DHE, each communication session generates new keys; unlike static keys. This type of key is called ephemeral and is a distinct characteristic of the DHE key exchange. Previous and future communications remain secure even if one session was hacked.

TLS 1.3 and Its Requirements

Published in August of 2018, TLS 1.3 addresses the weaknesses RSA key generation brings to internet encryption. One of the main requirements of TLS 1.3 is the mandated use of the DHE key exchange. The new standard no longer supports the RSA method. The goal of this is to protect past and future sessions by deploying perfect forward secrecy in the form of ephemeral keys.

Although the RSA algorithm can be used to create ephemeral keys, it requires more processing power than DHE. Since RSA key generation necessitates two large prime numbers, generating ephemeral keys using RSA is expensive in terms of processing power. Therefore, RSA is not ideal for use in perfect forward secrecy.

Also, RSA is not as secure as DHE simply due to the number of possible keys discrete logarithms can generate. In simple terms, the number matrix of viable keys in RSA is smaller than that in DHE. That means there are less numbers a hacker must sift through to crack an RSA key. If you compare an RSA key and a DHE key of the same size, DHE is more robust.

In addition to requiring DHE for key exchange, TLS 1.3 also offers some performance improvements. In this version of the standard, only one round trip is necessary to complete the initial handshake. Previously, TLS 1.2 required two round trips, which means 1.3 cuts encryption latency in half.

Lastly, TLS 1.3 remembers sites you have visited before and allows you to send encrypted data on the first message to the server. This is known as "zero round trip," or 0-RTT, and improves load times. Load times directly affect bounce rates of your site visitors. Google found that 53% of mobile site visitors leave a page that takes longer than three seconds to load.[4] So, by implementing TLS 1.3 standards, you have the opportunity to reduce bounce rates and convert visitors into purchasers.

4 "Find out how you stack up to new industry benchmarks for mobile page speed," Think with Google, February 2018, https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/

A Security Infrastructure Built For Encryption

While encrypting traffic is important, decrypting traffic is equally as important. Today, encryption-cloaked cyberattacks are common, and they are becoming more prevalent with each passing day. The number of encrypted attacks increased by 27% from 2017 to 2018.5 Methods of exposure to encrypted attacks include visiting phishing pages or compromised sites, visiting healthy pages with malvertising from infected ad servers, and receiving data containing encrypted malware. There are cases where attackers use SSL/TLS-based connections for command and control activities once an end-user is compromised. Because of this, enterprises need to decrypt the traffic coming into their network and investigate it for potential threats. Otherwise, they may be giving a free pass to encryption-cloaked malware, viruses, and more.

With the rising number of encrypted attacks and the requirement of ephemeral keys in TLS 1.3, there are new challenges for network monitoring. Previously, IT personnel could passively decrypt traffic by duplicating it, sending it to the decryption device, decrypting it, and forwarding it onto security and monitoring tools for analysis. This method is known as passive SSL decryption.

As shown in Figure 4 below, passive decryption in an out-of-band architecture is for inbound-only traffic monitoring scenarios and not does allow for the encryption/re-encryption of outbound communications.

5 "2019 Sonicwall Cyber Threat Report," Sonicwall, 2019, https://www.sonicwall.com/lp/2019-cyber-threat-report-lp/, pg. 31.

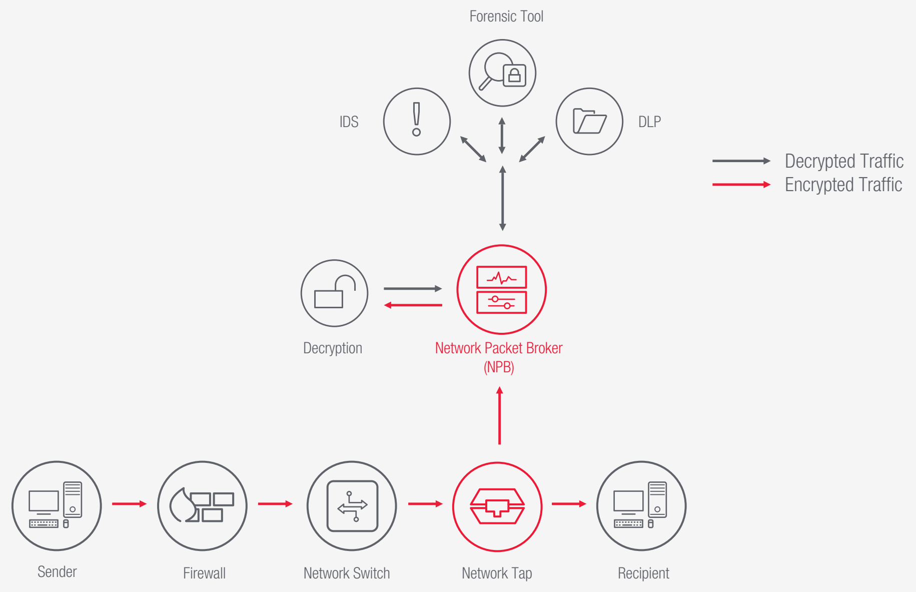

Figure 4. Passive SSL decryption scenario using a network packet broker

TLS 1.3 mandates the use of ephemeral keys, thus obsoleting the use of static keys and passive SSL. To use ephemeral keys, decryption must occur inline in the form of a transparent proxy. This decryption process, also called active SSL, is a known and trusted part of the network. It performs inline decryption for the end user recipients in a transparent fashion.

When using this scenario, decryption/encryption is supported in both directions (for inbound and outbound network traffic), as depicted in Figure 5. Once the cleartext analysis for security threats is complete, the good data is re-encrypted and sent to its destination within the network.

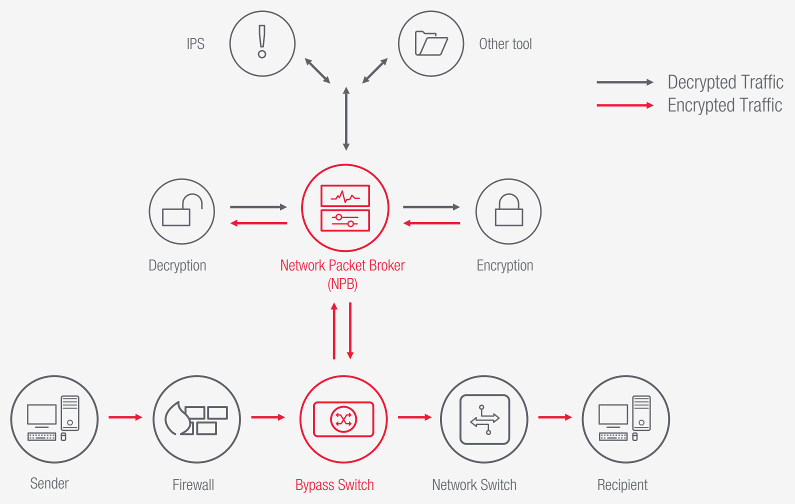

Figure 5. Active SSL decryption scenario using a network packet broker

Keysight's network packet brokers (NPBs) bring several security benefits to your network infrastructure, including visibility into encrypted data on the network and active SSL capabilities. They offer an integrated solution that is simpler and easier to use than decryption alternatives.

Keysight NPBs can:

- Decrypt and re-encrypt without degrading the performance of other functions such as traffic aggregation, filtering, grooming, replication, deduplication, and more
- Act as a central connection point between dozens of security tools and the traffic needed for inspection
- Load balance security and monitoring tools based on capacity
- Isolate and mask sensitive cleartext data — like email addresses, phone numbers, and health information — for privacy reasons
- Deliver a highly-resilient, fail-open system

The combination of these attributes allows Keysight NPBs to deliver a broad set of value-added features at a cost-effective price point.

Figure 6. Keysight's Vision X network packet broker built for inline and out-of-band monitoring architectures

Summary

A hacker will do anything to obtain and exploit your data and sensitive information. To combat their clever tactics and reduce vulnerability, organizations should deploy an active SSL decryption and encryption solution, like an NPB. Moreover, a solution that meets the latest standard requirements in TLS 1.3 —ephemeral key exchanges using DHE — is considered the most secure.

Learn more at: www.keysight.com

For more information on Keysight Technologies' products, applications or services, please contact your local Keysight office. The complete list is available at: www.keysight.com/find/contactus

KEYSIGHT TECHNOLOGIES