Cryptography in Hardware

Terence O'Brien, Ian Perry

***Abstract*—Since the beginning of recorded history, the act of procuring accurate information and utilizing it has marked the difference between those organizations which fail, and those which succeed. Nowhere is this more evident than in adverserial situations. The side which is able to learn the disposition of an enemy force while concealing their own is at a great advantage [1]. As history has progressed, the simple act of hiding information has ceased to be an effective option, and methods of obfuscating, or encrypting, that information has given birth to the field of cryptography. In the modern age, cryptography is used daily by the majority of the population through the usage of ecommerce, email, or any number of secured web sites. This ubiquitous usage has created a situation where quickly processing many crypotraphic requests at once is necessary, but by its nature, these requests are computationally expensive. This is where hardware extensions which implement cryptography come into play. This paper will explore the current state of cryptographic hardware, it's role in modern usage, and the required background information to understand it.**

***Keywords*—IEEE, Cryptography, Information Theory.**

I. Introduction

THE state of the modern world rests upon a strong foundation of information exchange. From shipping orders which bring just in time deliveries of food to stock market transactions signalling supply and demand across the world, the free flow of information is vital to our society. Just simply transferring information is important, but so is ensuring that no malicious actors have modified or deleted information to further their goals. To this effect, cryptography is used to ensure a few key points.

Non-repudiation is the idea that the author of a message is unable to state that they did not author said message. This can be seen as vital in many applications. Imagine if that stock broker with a buy order decides to disavow that purchase once he realizes the price has dropped significantly, or that certain messages are denied by an individual in a criminal investigation. The ability to guarantee that a message could only have been written by a specific individual is extremly important.

Keeping certain information secret is also of paramount importance. Once again, imagine that successful stock broker making a great monumental purchase having their buy information observed by an adversary, who could perhaps put in faster orders that are to their advantage, given this information.

Finally, ensuring that sent information remains intact without modification is of clear importance. For all these reasons, and some minor ones, the field of cryptography has been advanced. The solution to these problems is contained in four primary functions, two encryption techniques, hashing techniques, and random number generation. Due to the mathematical nature of these functions, and particularly for asymetric encryption, the computational toll can be significant. As they are ubiquitious in usage, a need for hardware acceleration is desired, and in some cases required.

The implementation of cryptography in hardware can be broken down into a few major components. Secure cryptoprocessors perform the previously mentioned functions in a secure, physically tamper resistent environment, without providing acceleration. Special modules, used to build SSL accelerators, actually perform acceleration of these functions, but ordinarrily do not provide such tamper resistence. Hardware security modules roll all of these components into one, and include random number generators and hashing hardware as well. The importance of securely implementing these functions in hardware, and software as well, can not be understated, and will be explored thoroughly in this paper.

II. History of Cryptography

Cryptography and cryptanalysis, up until World War II, didn't involve any mechanical or electromechanical tools. By World War II, cipher machines were widely used. The invention and subsequent breaking of these tools was one of the greatest breakthroughs in cryptography for thousands of years up to that point due to its impact in global affairs. To defeat another form of German cryptography, known as the Fish ciphers, members of the Cipher Bureau designed the world's first programmable digital electronic computer, the Colossus. The Americans used the SIGABA, an electromechanical rotor design similar to the Enigma but with major improvements making it undefeated as far as anyone knows in World War II.

First described in 1882, the one-time pad is the only form of encryption that cannot be cracked. It requires the use of a pre-shared key that can only be used once and has to be at least the length of the message to be encrypted. Each character or bit of the message is encrypted by combining it with the corresponding character or bit from the key using modular addition. This has been proven to be unbreakable. If an adversary intercepts the ciphertext, there is no way to deduce the original message. That ciphertext can be decrypted to form any message just by varying the key. This form of encryption hasn't found mainstream adoption because it is very difficult to ensure the key material is completely random, only used once, never gets discovered, and is destroyed after use. In machine implementations, the XOR function is often used to combine the plaintext and key elements because it usually has a native machine instruction and is therefore fast. At the end of World War II the United States managed to solve a one-time pad system used by the German Foreign Office for its important messages. The keys that were used were insecure because they weren't completely random. The machine used to generate the "seemingly" random keys actually produced predictable output.

The introduction of symmetric key ciphers after World War II marked one of the most notable advances in cryptography. With the widespread adoption of the Internet came the need for a widespread standard of encryption. Data Encryption Standard (DES) was the first to be adopted, soon after it was replaced by the Advanced Encryption Standard (AES). Finally with the advent of the Secure Socket Layer (SSL) all transactions done on the Internet were protected by encryption.

The problem with crypto systems leading up to asymmetric key encryption, or public-key ciphers, was the necessity for the sender and receiver to share the same key while at the same time keeping it secret. This led to a problem with key distribution. In the early days of cryptography the keys were exchanged by trustworthy couriers, face-to-face contact, or even a loyal carrier pigeon. In the age of the Internet where all communication required keys shared on a two-person basis, this problem of distribution became hard to manage. Asymmetric key encryption works with pairs of mathematically related keys, private and public. Since the public key could be the one that needed to be exchanged, no secure channels were needed for the exchange. As long as your private key remained private, encrypted data could only be assessed by brute force.

III. Cryptography Basics

A review of the basics of cryptography as a whole is beyond the scope of this paper. To do that topic justice would require an entire paper in itself. Thus, the basics of modern cryptography as it applies to an understanding of its implementation in hardware will be presented.

A. Symmetric Key Encryption

Modern cryptography hinges on the use of four main functions, symmetric encryption, asymmetric encryption, hashing, and random number generation. Symmetric encryption is a method most often thought of when encryption is mentioned. It is the use of a single key to both encrypt and decrypt a message. Both user's must have the same key, and it must be shared in some fashion. This has classically been a limiting factor in the adoption and usage of cryptography.

In order to ensure a key has not been lost to the adversary, key sets must be periodically changed, which entails distributing entire new sets of keys, or having large tomes with predetermined keys available. This is an enormous burden which grows roughly on the order of n^2, where n is the number of users. Not only is this cost prohibitive at a certain n, but the logistics of securely transffering key material precludes it's use, particularly in the case of warfare or espionage.

Nevertheless, symmetric key encryption has the advantage of being relatively less computationally expensive than its asymmetric counterpart. This results in many uses where the initial secure session is created through the use of asymmetric methods, while the bulk of the data remains secure through symmetric encryption.

B. Asymmetric Key Encryption

Symmetric encryption is extremely fast and therefore perfect for encrypting traffic on the Internet. The only issue with this is distributing the single symmetric key. If the key is sent in plaintext to the person being communicated with, any eavesdropper on the Internet could see it and decrypt every message sent after that. This is why asymmetric key encryption is so important. Although it is nowhere near as performant as symmetric key, it's ability to *begin* an encrypted communication session is what gives it its power. With asymmetric the key that gets sent over in plaintext is what's known as the public key. It, in combination, with each users' private keys is what performs the encryption like in Fig. **??**. Through this means, the far faster method of symmetric encryption can be began without an eavesdropper ever having the ability to eavesdrop.

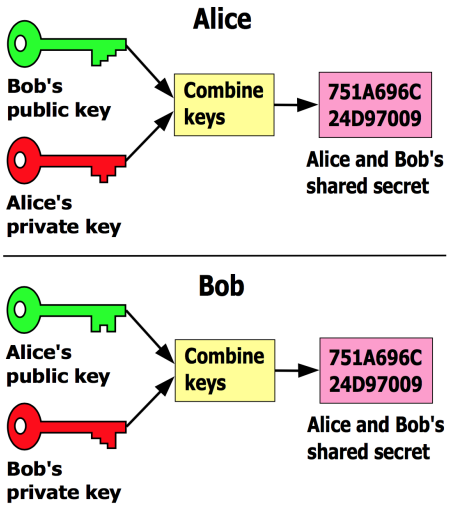

Fig. 1: Public key encryption [2]

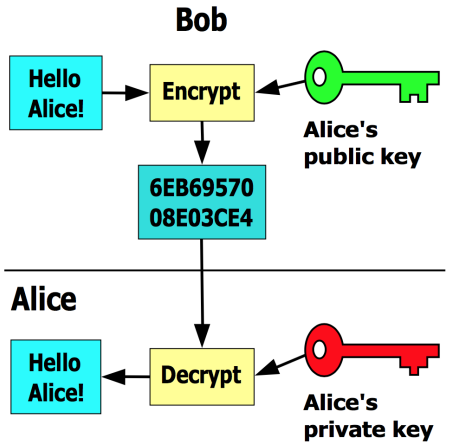

Fig. 2: Encrypting and Decrypting with Asymmetric Keys [2]

The secure exchange of two people's communication starts with the swapping of public keys. The first person encrypts a message using their private key followed by the other's public key. That encrypted message is then sent to and decrypted

by the other person with their private key followed by the first's public key. If the resulting message is recognizable, both parties can assume the communication is secure. Fig. 2 illustrates this exchange.

C. Hashing Functions

Hashing functions are ubiquitous in the world of computing. Whether they are used in common data structures such as a hash map, caching, computer graphics [3], or cryptography, hash functions are used in many domains for many functions.

A hash function takes a variable length input and transforms it into a fixed length output. This transformation is one way, and must not be able to be reversed. Three primary uses for hashing are found in cryptography, they are used in password checking, message authentication codes [4], and file content digesting.

When a user logs into a domain workstation, the password is not sent over the wire for authentication. Instead, it is hashed, and that hash is compared with the stored hash. This prevents a plaint text, or even an encrypted, password from being sent, and possibly being intercepted. Further, when storing passwords, the accepted method of doing so is by hashing them, as well as salting them.

In asymmetric encryption, hashing is used to check the integrity of messages, the so called authentication code. By performing a hash of data to be encrypted, and then encrypting that hash with a private key, the recipient can decrypt the hash, verify that the data matches the hash, and then know that the user who created the hash truly sent the message. A similar idea is used in file content digesting. By storing the hash of a file in a public place, any user which downloads that file can perform a hash on their machine, compare it to the public hash, and ensure that the software they have downloaded is not a fake copy or has been altered in some way.

There are many hashing algorithms, but the most popular are MD5, SHA1, and SHA2, which is broken up into SHA-256 and SHA-512. In recent times, MD5 has become less secure [6] [5], and SHA1 is beginning to succumb to Moore's law as well. In the very near future, SHA2 varieties will make up the bulk of cryptographic hashing.

D. Random Number Generation

One integral challenge a computer system has to achieve when doing anything with cryptography is creating truly random data. This is extremely important, because as discussed above, cryptographic systems have been defeated simply because the random generator used to create a key wasn't random enough. This is a difficult challenge due to the nature of digital machines always being in well-defined, very predictable states, only changing when programs tell it to. The best that machines like this can do is simulate randomness through algorithms that create pseudorandom numbers following mathematical procedures. Such a set of data would look very random, but another computer following the same procedure could create the exact same sequence. These pseudorandom numbers usually start off with a special seed value otherwise they'd always generate the same numbers.

Luckily, digital machines can look outward to the vast, chaotic universe around them for pure randomness. One such tool that took advantage of such chaos was called Lavarand and consisted of a lava lamp with a camera. The camera would take pictures of the lava lamp's seemingly random blob-like movements and through computer visions algorithms output random numbers. Such a device could then just be used to create a random seed for a pseudorandom number generator that could generate random numbers at a much higher frequency. This example shows that randomness exists in nature and can be used for digital purposes, but is obviously limited.

E. Cryptographic Protocols

Cryptographic protocols are security minded protocols which wrap the various cryptographic algorithms into cohesive blocks which perform all of the steps of initializing a session, encrypting, and decrypting. Usability is of paramount importance and has been the driving factor for adoption of cryptography by common users. Although there are many such protocols, for the purposes of this paper, only SSL/TLS and HTTPS will be discussed, as they encompass the bulk of the need for hardware acceleration.

SSL/TLS, or Secure Sockets Layer/Transport Layer Security, often just called SSL, is a protocol which provides secure communications for VoIP, IM and web browsers. When using a web browser to connect to a server over HTTPS, it is nothing more than an HTTP session over an SSL channel [7]. One of the key elements of this protocol is the SSL certificate. Certificate Authorities (CA) sign certificates for domains and store their information. When a server provides a certificate, it can be vetted against the information in the CA to ensure that the server is who it claims to be. A great deal of trust and security is implicit in the operation of a CA, and thus has been theorized as a potential weak point by public key critics. To date no serious breaches of a CA's operation have occurred.

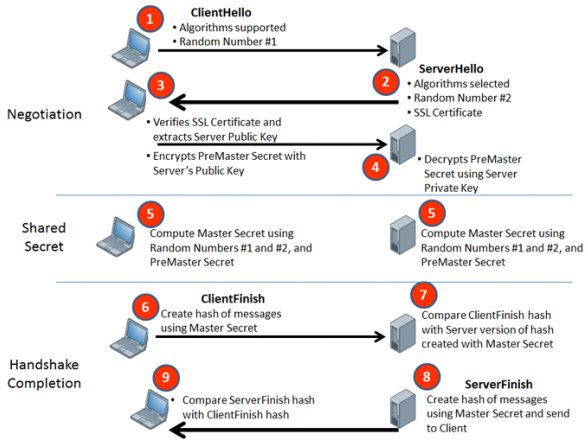

Fig. 3: SSL Process Diagram [9]

This handshake, as the initial authentication is called, also contains information regarding the remainder of the session.

Included are details such as the TLS version, and which cipher suite will be used. Available suites include AES, Camellia, and SEED [8]. Available protocols for handshakes include RSA, PSK, various forms of Diffie-Helman, and others. Public-key infrastructure, which implies asymmetric keys, is used for this handshake. Further, this key protects the transmission of the Master Secret, typically an AES key, which is then in turn used to protect the actual data. This process of asymmetric keys allowing the transmission of symmetric keys is the realization of overcoming the obstacle of symmetric keys. With a method of authenticating user identity, and then passing the encryption key, all of the logistical cost discussed earlier is negated. Further, the computational cost of encrypting via AES and symmetric keys is must lower than that of performing asymmetric operations. The Fig. 3 demonstrates the operation of SSL in a more detailed manner.

IV. Standalone Cryptographic Hardware

In the course of using an e-commerce web page, a user may only require a single SSL session every few minutes. This level of computation is quite low, and does not require much processing power. Contrast this with the number of SSL connections per second that even a mid or low activity server require to service all incoming connections, and it is clear that hardware acceleration is required to alleviate the burden. To this effect the past two decades have seen a niche market arise to provide cryptographic accelerators, as well as entire cryptographic hardware suites which take care of key management as well as algorithmic heavy lifting.

A. Secure Cryptoprocessors

Secure Cryptoprocessors form the heart of many security based devices. The Primary focus of these processors is not acceleration or performance, but of performing cryptographic operations in a single tamper proof location [10]. When utilizing encryption in any form, it is common for one end of the session, if not both, to be in a non-secure location. An example of this could be as mundane as a desktop sitting in an internet cafe. Without explicit guardianship of the physical device, it is conceivable that an adversary would be able to gain access to the hardware in order to either extract key information, or modify it in such a way as to produce the information at a later time. If this key material were to be lost to the adversary, impersonation of the user, and the decrypting of communications for both sender and receiver could occur. There is no software or algorithm that could prevent this from occurring, as is encapsulated in the The purpose of a secure cryptoprocessor is to alleviate this concern.

In Fig. 4 a secure cryptoprocessor block diagram can be seen. Although this particular processor is from a theoretical white paper, it will serve to demonstrate operations. As can be expected, scant details are the norm regarding anything to do with security, especially so with cryptography. At the periphery of the block diagram is the physical security boundary. This construct is common to all cryptoprocessors and is the demarcation point where no unencrypted information is allowed to pass. All keys, all processing, must occur and be

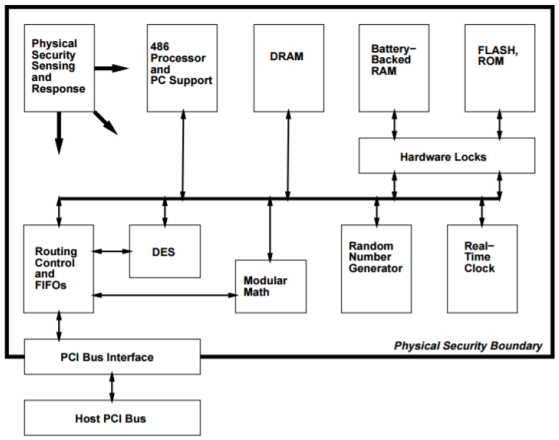

Fig. 4: Secure Cryptoprocessor Diagram [10]

stored behind that boundary, unless it has been encrypted. This boundary is not only figurative, it is a literal physical boundary. The processor is enmeshed in a conductive grid [10] which allows any physical tampering to be detected. In the event that tampering is detected, the key material, at the very least, is deleted by zeroizing the memory. Ideally the operating system is also zeroized along with any other information. In this way, it can be assured that key material on the chip stays on the chip, and can not be removed, at least in a physical manner.

B. Random Number Generator

The implementation of a random number generator is of paramount importance for cryptographic purposes. For this reason researchers designed special hardware to go in Intel chips that was power efficient and obeyed common cryptographic standards [16].

The design consists only of two transistors and two inverters and can be seen in Fig. 5. Typically in digital systems the goal is to have only logic 1's and 0's. However when designing a random number generator, teetering that can sometimes happen in digital logic is a good thing and can even be exploited. When the clock in this design hits and the two transistors allow nodes A and B to be at a logic 1, the inverters are fighting to determine which is going to invert from a 0 to a 1 or from a 1 to a 0. This fighting is called metastability and always ends in a random winner determined by such chaotic things as thermal noise and random atomic vibrations.

Just one of these digital circuits wouldn't be enough. Small variations in the fabrication process can cause differences in the inverters and even lead to either 1 or 0 being favored. To circumvent such a problem, designers built a feedback loop into the hardware that ensures the two possible outcomes occur roughly equal amounts of the time.

Most pseudorandom number generation algorithms require the initial seed be something like a 256-bit number. Intel's design goes a step further than just sampling their inverter logic 256 times, it instead samples 512-bits and performs a

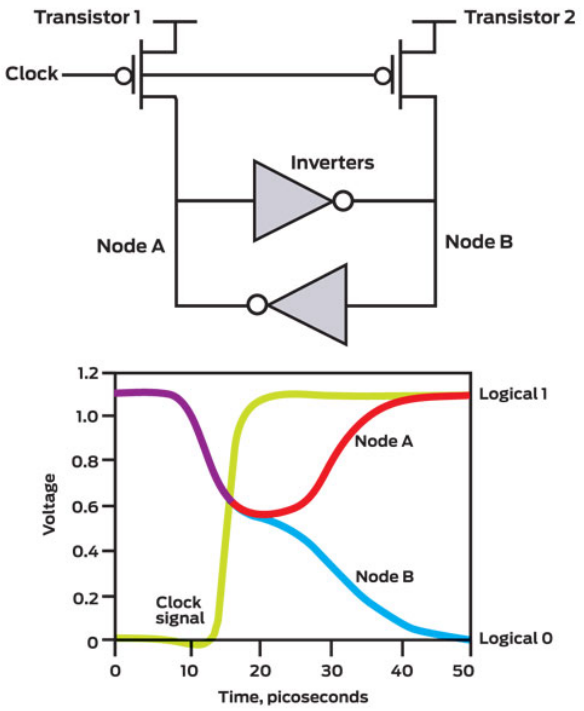

Fig. 5: Random Number Generator Hardware [16]

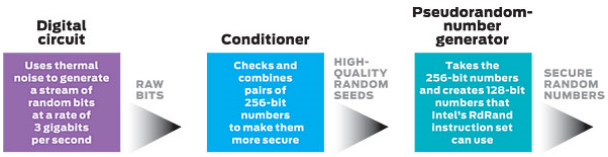

Fig. 6: Random Number Generator Hardware [16]

cryptographic operation combining both 256-bit halves into one very random 256-bit number that can finally be used in a proven pseudorandom number generator. Finally this algorithm can produce random number at very high frequencies. The instruction RdRand was introduced with this hardware to access 16-, 32-, or 64-bit random values from the Intel chips. A diagram of this process may be seen in Fig. 6.

C. SSL Acceleration

In recent years, there has been a push to shift all web pages with a login to only allow HTTPS connections. Not just for e-commerce or banking pages, but for email and even social media. Google has begun to incentivize web page owners to implement HTTPS [11] by favoring pages with implemented SSL in its search page rankings. Further, all e-commerce is conducted over HTTPS. What this amounts to is that large web servers must handle many thousands of SSL sessions per second. As asymmetric encryption is much more expensive computationally, hardware which can offload this processing is invaluable. As of 2010, a typical CPU core would only be capable of 1,000 HTTPS transactions per second, while the same core would be capable of handling 10,000 HTTP transactions per second [12].

To this effect, SSL Accelerators were created to perform just such a function. The majority of the computation required for SSL, particularly the asymmetric handshake procedure, is composed of integer arithmetic such as n-bit multiplication, where n is the number of bits in the key [12]. This lends itself quite readily to hardware acceleration with specialized hardware implementing many such operations performed in serial. Theoretical performance can approach O(k) time, where k is the number of bits in two integers [12]. Due to the linear nature of symmetric algorithms, this parallelization is not possible, and thus most hardware acceleration occurs in the realm of asymmetric encryption.

Several manufacturers produce SSL accelerators, which are designed for server applications, with Sun being the leader among them. In 2002 the state of the art Crypto Accelerator 1000 PCI card was capable of 4300 new transactions per second, while by 2013 the price had dropped to a tenth, and performance increased to 13,000 new transactions per second, along with a concurrent increase in maximum key length.

A typical processor used in these applications would be the BCM5821 [13], a Broadcom e-commerce chip, which at the time was capable of processing 4000 1024-bit RSA transactions a second. It operated at 125-MHz and was based on 0.18m CMOS technology, with a low power consumption of 2.8 Watts.

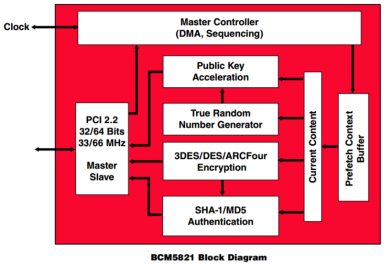

Fig. 7: BCM5821 Block Diagram [13]

In Fig. 7, a block diagram of the BCM5821 can be seen. The four common cryptographic components can be seen in the diagram, which include a SHA/MD5 hashing block, a random number generator, a symmetric encryption accelerator, which is the 3DES/DES/ARCFour block, and an asymmetric accelerator, named the public key acceleration block.

D. Hardware Security Modules

The need for physical security and hardware acceleration have been combined into a single discrete network component, the hardware security module (HSM). In addition to these two functions, many

V. Instruction Set Extensions

A. AES-NI

In 2010 Intel introduced AES-NI into the Intel Core processor family [14] [15]. AES-NI stands for Advanced Encryption Standard - New Instructions. These six new instructions were implemented in hardware such that they accelerated the more compute intensive steps of AES. Two of these instructions aided in the acceleration of encrypting, two helped decrypting, and the last two improved the generation of keys. These instructions don't just improve performance, they improve security as well. By not requiring software writers to implement the key stages of AES, small security flaws can't be introduced into the system due to incorrect implementations. Software implementations using tables also sometimes fall victim to major timing and cache-based attacks because a lot of the data needed to run the algorithms is available in software and can be accessed maliciously. Having AES run completely in hardware defeats these vulnerabilities. These new instructions can also speedup the performance over the completely software approaches by 3 to 10 times.

B. Intel SHA Extensions

VI. Conclusion

Ever since the dawn of humanity, the very nature of communication included those that wished to communicate, and those that wished to eavesdrop. In the early stages this took the form of simple mathematical heuristics that could disguise the message you wished to communicate into seemingly random gibberish. As the stakes were raised with more widespread conflict, so did the stakes of protecting communications as well as breaking other's. With the advent of computers suddenly a machine could crunch far more complex algorithms than a human ever could. This allowed not only governments to have the strongest encryption, but the entire population. This new ability gave rise to higher demand for performance and efficiency. Now specialized hardware in every devise can generate completely random numbers to drive completely secure cryptographic systems. The internet is designed to allow every person in the world to communicate with one another and cryptography is designed to protect that communication.

References

[1] S. Tzu, "A quote from the art of war," in Good Reads, Goodreads, 2015. [Online]. Available: http://www.goodreads.com/quotes/744436-conceal-your-dispositions-and-your-condition-will-remain-secret-which. Accessed: Dec. 13, 2016.

[2] ict@innovation, "File: Ict-innovation-LPI-Fig-110-3 1.png - Wikimedia commons," in Wikipedia, 2012. [Online]. Available: https://commons.wikimedia.org/wiki/File:Ict-innovation-LPI-Fig-110-31.png. Accessed: Dec. 13, 2016.

[3] G. J. van den Braak, J. Gmez-Luna, J. M. Gonzlez-Linares, H. Corporaal and N. Guil, "Configurable XOR Hash Functions for Banked Scratchpad Memories in GPUs," in IEEE Transactions on Computers, vol. 65, no. 7, pp. 2045-2058, July 1 2016. doi: 10.1109/TC.2015.2479595

[4] P. Gutmann, D. Naccache and C. C. Palmer, "When hashes collide [applied cryptography]," in IEEE Security & Privacy, vol. 3, no. 3, pp. 68-71, May-June 2005. doi: 10.1109/MSP.2005.84

[5] J. Anish Dev, "Usage of botnets for high speed MD5 hash cracking," Third International Conference on Innovative Computing Technology (INTECH 2013), London, 2013, pp. 314-320. doi: 10.1109/INTECH.2013.6653658

[6] H. Kumar et al., "Rainbow table to crack password using MD5 hashing algorithm," 2013 IEEE Conference on Information & Communication Technologies, JeJu Island, 2013, pp. 433-439. doi: 10.1109/CICT.2013.6558135

[7] 2016 S. Corporation, "SSL by Symantec - learn how SSL works," in Symantec, 1995. [Online]. Available: https://www.symantec.com/content/en/us/enterprise/white_papers/b-beginners-guide-to-ssl-certificates_WP.pdf. Accessed: Dec. 14, 2016.

[8] J. Salowey and A. Choudhury, "AES Galois Counter Mode (GCM) Cipher Suites for TLS," RFC 5288, Aug. 2008.

[9] "What is SSL?," in IdenTrustSSL. [Online]. Available: https://www.identrustssl.com/images/learn_ssl_diagram.gif. Accessed: Dec. 14, 2016.

[10] "Building a high-performance, programmable secure coprocessor," Comput. Netw., vol. 31, no. 9, pp. 831'860, Apr. 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=324119.324128

[11] D. Goodin, "In major shift, Google boosts search rankings of HTTPS-protected sites," Ars Technica, 2014. [Online]. Available: http://arstechnica.com/security/2014/08/in-major-shift-google-boosts-search-rankings-of-https-protected-sites/. Accessed: Dec. 14, 2016.

[12] K. Jang and S. Han et al., "Accelerating SSL with GPUs," ACM SIGCOMM Computer Communication Review, vol. 41, issue 1, January 2011

[13] Broadcom, "BCM5821," in Elcodis. [Online]. Available: http://datasheet.elcodis.com/pdf2/71/46/714600/bcm5825a1kpbg.pdf. Accessed: Dec. 14, 2016.

[14] R. C., "AES-NI in Laymen's Terms," in Intel, 2012. [Online]. Available: https://software.intel.com/en-us/blogs/2012/01/11/aes-ni-in-laymens-terms. Accessed: Dec. 14, 2016.

[15] J. Rott, "Intel Advanced Encryption Standard Instructions (AES-NI)," in Intel, 2012. [Online]. Available: https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni. Accessed: Dec. 14, 2016.

[16] G. Taylor and G. Cox, "Behind Intel"s new random-number generator," in IEEE, IEEE Spectrum: Technology, Engineering, and Science News, 2011. [Online]. Available: http://spectrum.ieee.org/computing/hardware/behind-intels-new-randomnumber-generator. Accessed: Dec. 14, 2016.