Applications of Abstract Algebraic Structures in Semantically Secure Encryption Extending to Network Security

Jacob W. Adkins
Kent State University at Stark, Ohio, USA
jadkin31@kent.edu

2/1/15

Abstract. I present research on various encryption schemes and their applications in securing data. To do this, I discuss their vulnerabilities, implementations (both logically and mathematically), and briefly mention their durability against cryptanalysis attacks. I will discuss their mathematical properties, and theories that are assumed to hold within a security parameter. Namely the RSA (Rivest, Shamir, and Adelman) Encryption Algorithm and AES (Advanced Encryption Standard) Algorithm in CBC mode (Cipher Block Chaining) against chosen-plaintext attacks[1]. Finally, I will explain why all cryptography will eventually converge to a field of quantum mechanics.

Key words. Public-key encryption, Symmetric-key encryption, RSA, AES, DES, CBC.

I. Introduction

The Internet[2] has changed many of the ways we interact with each other on a daily basis. From commerce to communication, many interactions are no longer preformed face-to-face. It then becomes obvious as to why one may wish to keep some information confidential; indeed, secrecy is of the most critical issues security professionals deal with on a daily basis. While security protocols are deployed on every level of a network, this secrecy tends to rely on the Secure Socket Layer (SSL), which can be found somewhere in between the Transport Layer and Application Layer[3]. One of the services the SSL delivers is a process known as encryption. Encryption uses a so-called key to manipulate any given message beyond recognition so that only the intended party can read the message. More specifically, encryption takes a piece of plaintext[4] and converts it into ciphertext[5]. This must be done in a way such that it cannot be easily reversed without a specific piece of information, in other words, a key. This mathematical concept is known as *trapdoor function*. This, among other mathematical properties must hold in order for the data to remain secure.

In addition to these properties, there are two well-known fundamentals of encryption. The first one is that a message must contain some *Redundancy* [1]. Certainly, in order for a message

[1] A situation in which an attacker has access to limited plaintext and its corresponding ciphertext.
[2] I refer to the World Wide Web instead of a generic internetwork with a capital 'I'.
[3] While vague, this refers to the OSI reference model which is not commonly used in practice. The actual location of the SSL can very.
[4] A message in linguistics.
[5] The output of an encryption algorithm.

to be kept a secret, it will require some information that is not essential to the original message. The second fundamental of encryption is that a message must contain *freshness*. [1] This will prevent replay attacks[6]. Furthermore, there exists another convention known as *Kerckhoff's Principle* which explicitly states, "All algorithms must be public; only the keys[7] are kept secret," [1]. The reasoning behind this proposition proves to be axiomatic. One would not want to assume that no one else knows the implementation of their algorithm when, in fact, they do. In other words, security by ambiguity is not a valid solution. In fact, all of the applied algorithms are widely published, and well known. It is also worth mentioning that the actual size of a key is directly dependent on the algorithm, and can range anywhere from eighty to 4096 bits.

History The first documented instance of crypto appears during the year 1900 BC within "non-traditional" Egyptian hieroglyphics [3]. Although, the most prominent construct occurs some time later in 49 BC during the rule of Julius Caesar. Caesar used what is now known as a substitution cipher, also known as the Caesar Cipher, to share classified information secretly. The United States government also applied the language of the native Navajo Indians to combat the Japanese in World War II. This language is tonal and complex, and being able to communicate covertly played a large part in their victory [1].

Today, we say that encryption of information is to be computationally hard, and decryption, relatively easy; however, this was not always true. Early cryptographic methods applied the inverse of this ideology. The need for the latter convention resulted in what are now called *cryptographic primitives* [1]. These include the substitution cipher, the transposition cipher, and the product cipher. These are the first structures that rely on a cipher[8] instead of a code. The absence of injected data in a cipher minimizes its redundancy, yielding great application for transmission through an internet.

Due to the computational capacity available to users today, or even several years after the development of the ARPANET, these primitives are no longer secure in their fundamental form; however, there exists a construction in which these cryptographic primitives can be used to effectively secure data today. This construction is called AES (Advanced Encryption Standard). Additionally, it is fortified into a scheme when run in CBC (Cipher Block Chaining) mode. Originally developed by IBM, and controversially adopted by the NSA as the standard for unclassified information, AES is used to securely move data as of May 26th, 2002 [4]; however, the algorithm originates from its predecessor, Triple DES (Data Encryption Standard) which is derived from DES. DES was standardized by NIST in July of 1977 [5]. Indeed this group of symmetric key algorithms has gone through a series of changes to combat the increasing computational capabilities of cryptanalysis adversaries.

It is not always necessary to edit the implementation of the protocols themselves, more frequently we simply change the key sizes used to protect the data. By Moore's Law[9], it will

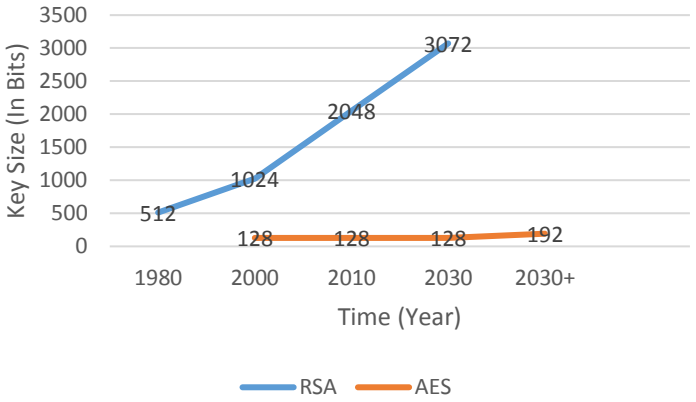
[6] An attack on a server that involves sending old packets that contain commands that are old (they have already been processed), but valid.

[7] Typically a large number that has certain properties dependent upon the algorithm.

[8] The encryption of an n-bit plaintext yields a n-bit output. Compared to a code which may change the length of the ciphertext.

[9] "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will remain nearly constant for at least 10 years," [6]. With this assumption, the computational capacity then grows at a relative rate, yielding information about acceptable key sizes.

become considerably easier for an attacker to break a cipher after a projected amount of time; consequently, we must increase the complexity of the input. Figure 1 delineates the comparison between the two methods.



Key Size Increases over Time

Figure 1. Shows key size increase over time. RSA data has been inferred from Table 4 in NIST Special Publication 800-57 *Recommendation for Key Management – Part 1: General (Revised)*. AES data originates from the same document, Table 2. Notice, these are instant jumps, not steady increase. The lines are to show a comparison between growth times [7].

Modern Determinate While there are many encryption algorithms, the most widely used are the public/private key algorithm and the symmetric key algorithm. Although both solutions will achieve the same end, all encryption must rely on an assumed computational hardness[10] of a theoretical problem, of which can differ. Namely, we assume that no algorithm can run in *polynomial time*[11] that is capable of breaking the cipher[12]. Currently, the lower bounds[13] of an algorithm cannot be thoroughly tested; therefore, security is based on unproven, so-called, *cryptographic hardness assumptions*. [2] By design, these are statistically impossible to break, as we will see, and indeed over a substantially long period of time, they have held[14]. Additionally, it is worth noting that an unbreakable cipher exists, along with a proven hardness assumption, and key sharing mechanism; however, currently it is too expensive to implement.

Details of My Assembly I begin by introducing the reader to some basic, yet foreign (to those not familiar with abstract algebra), mathematical concepts and assumptions. I then go on to use these preliminaries to describe why and how encryption works. After doing so, I explain the circumstances an algorithm may fail. Finally, I conclude by describing quantum cryptography and its application in the future of the field.

---

[10] In computational complexity, *hard* typically refers to the average case; however, I am talking about the longest (most complex) path through an algorithm.

[11] An algorithm is said to be solvable in *polynomial time* if the number of steps required to complete the algorithm for a given input is $O(n^k)$ for some non-negetive integer k where n is the complexity of the input; according to [8].

[12] Undesired decryption (inverse operation of encryption) without the granted use of the key.

[13] The $O(n)$ of a particular algorithm must be proven to have the lowest computational complexity that exists, has existed, or ever will exist in the future for solving a given problem.

[14] This is known as a heuristic proof.

Purpose of My Composition In this paper, I delineate the design and implementation of one-way functions, and the assumptions necessary for a structure to be considered *secure*. In doing so, I aim to guide introductory level scholars through the most basic research required to explore more advanced topics in provable encryption schemes. Ultimately, this will allow others to contribute to the field of study considerably easier allowing us to advance the field of Network Security more quickly.

II. Preliminaries

2.1 *Abstract Algebraic Structures and Theoretic Constructions of Pseudorandom Objects*

Lemma 1. (Probabilistic Polynomial Time) $An\ efficient\ computation\ carried\ out\ within\ polynomial\ time\ is\ said\ to\ be\ run\ in\ PPT$ [11, Ch. 3.1.2, pp. 54].

Lemma 2. (Random Oracle Schemes) $Supplies\ an\ instance\ of\ a\ quintessential\ hash\ function$[15]$;\ that\ is,\ \hbar = X \mapsto Y\ which\ is\ chosen\ randomly\ from\ \mathcal{F}^{X,Y},\ and\ only\ oracle\ access\ is\ given\ to\ the\ function.\ This\ means\ that\ we\ are\ not\ given\ an\ algorithm\ or\ formula\ for\ computation.\ In\ order\ to\ recieve\ the\ output\ of\ the\ function\ we\ must$ "query the oracle". $Ultimately,\ this\ acts\ like\ a\ reference\ of\ random\ and\ uniform\ data.\ Of\ course\ these\ are\ not\ physical\ structures;\ although,\ the\ idea\ is\ that\ a\ well\ built\ hash\ function\ will$ "behave" $in\ this\ convention.\ Cryptosystems\ that\ require\ the\ use\ of\ an\ oracle\ are\ said\ to\ be\ heuristically\ secure\ within\ that\ model$ [10, Ch. 4.2.1, pp. 122] and [2].

Definition 1 (Negligible Function). $When\ a\ function\ has\ a\ probability\ of\ success\ that\ is\ smaller\ than\ the\ reciprocal\ of\ any\ polynomial\ function,\ it\ is\ said\ to\ be\ a\ Negligible\ Function\ \eta(k).\ That\ is,\ if\ \exists_N \ni \{n : n \in \mathbb{Z}\ and\ n > N\}\ then\ for\ every\ PPT\ function\ p(\cdot)$:

$$\eta(n) := \frac{1}{p(n)} = \frac{1}{superpoly(k)} > f(n)$$

[11, Ch. 3.2.1, pp. 56].

Definition 2 (One-Way Function). $A\ theoretical\ function\ f : \{0,1\}^* \to \{0,1\}^*\ is\ called$ one-way $if\ the\ following\ two\ conditions\ hold:$

1. Easy to compute: $There\ exists\ a\ polynomial - time\ algorithm\ M_f\ such\ that\ on\ input\ any\ x \in \{0,1\}^*, M_f\ outputs\ f(x)\ (M_f(x) = f(x)\ \forall_x).$
2. Hard to invert: $For\ every\ PPT\ inverting\ algorithm\ \mathcal{A},\ there\ exists\ a\ negligible\ function\ \eta\ such\ that$

$$\Pr\left[\mathcal{A}\left(f(x)\right) \in f^{-1}\left(f(x)\right)\right] \leq \eta(n)$$

[15] The problem with traditional functions is that given a very small amount of data ((x_1, y_1) and (x_2, y_2)), one can calculate the output of the function at other points without actually having to evaluate the function at them.

where the probability is taken over the uniform [*and independently chosen*] *choice of* x *in* $\{0,1\}^n$ *and the randomness of* $\mathcal{A}$.

It is only guaranteed that a one-way function is hard to invert when the input is *uniformly distributed.* Thus, there may be many inputs (albeit a negligible fraction) for which the function can be inverted. Furthermore, as usual for asymptotic definitions, it is only guaranteed that it be hard to invert for all long enough values of x[16], as equivalently demonstrated in [11, Sec. 6.1, Def. 1].

Definition 2 (One-time pad). *Let the one − time pad,* $\prod = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be defined as an encryption scheme for which the message space* $\mathcal{M}$ *is as large as the key space* $\mathcal{K}$*, and a message* m*, and key,* k*, have an equal length. Then* $\mathsf{Enc} := C = k \oplus m$*, and* $\mathsf{Dec} := m = C \oplus k$*. Additionally we allow* Gen *to assign, on input* $1^{|m|}$, $k \leftarrow \{0,1\}^{|m|}$*, where* $|m|$ *denotes the length of the message. This cipher is said to be unbreakable because with any key, any message of the same length is a possible result* [1].

2.2 *Cryptographic Hardness Assumptions*

Lemma 1 (Indistinguishable under Chosen-Plaintext Attack). *For a cryptosystem to be considered secure in terms of* $IND - CPA$ *(Indistinguishability under Chosen − Plaintext Attack), an adversary must not be able to distinguish between two messages of equal length and the ciphertext of one of those messages with probability greater than one half. We call the process in which we determine an adversary's advantage*[17] *over a cryptosystem, a game* [13].

Definition 1 (Discrete Logarithm). *Let there be an instance with a multiplicative group*[18] $(G,\cdot)$*, an element* $\propto \in G$ *having order* n*, and an element* $\beta \epsilon <\propto>$.

The problem domain then requires the discovery of a unique integer a*,* $0 \leq a \leq n-1$*, such that* $\propto^{\alpha} = \beta$.

The discrete logarithm of β *can then be denoted by* $a = log_{\propto}\beta$ [10, Sec. 6.1, Prob. 6.1].

Definition 2 (Galois Field Theory). *Let* $GF(p^m)$ *define a finite field, namely an extension field where* p *is an element of* $\prod_{Prime} \mathbb{Z}$ *and* $m \in \mathbb{Z}^+$*. Within this field, all elements are defined by polynomials with coefficeints determined by their binary existence in the prime subfield* $GF(m)$*. These fields then have all operations of the ordinary field,*

[16] The converse of this statement does not hold. A function that is not one-way is not necessarily easy to invert. Additionally, the existence of this function is inherently an assumption about *computational complexity* and *computational hardness*. In other words, it can be solved in theory, but not efficiently in practice. That is, it can be solved in exponential time. [intro to modern crypto]

[17] An adversary is said to have an advantage over a cryptosystem if he is able to distinguish between a message pair and a ciphertext of one of the pair.

[18] See *Group Theory/Cyclic Groups*.

$\mathbb{R}$.

III. Implementation of the Cryptosystems

3.1 *RSA Public/Private Key Scheme*

In this section I aim to explain the algorithms from a pure point of logic so that it will be easier to understand the algorithms, possibly the preliminaries, and also the remainder of the paper. Public Key Crypto employs the use of not only one key, as you might think, but two. One key is said to be private, and the other one is public. The names for these keys will prove to be axiomatic. You can think of the information as the lock that requires two different keys. One key will be able only to *unlock* the data (the private key), and the other will only be able to *lock* the data (the public key).

The first step in the algorithm is the generation of the keys. There are standards and recommendations for the specific methods one should use to go about their generation and the implementation of the algorithm as a whole called PKCS, public-key cryptography standards. For now, it is sufficient to mention that they exist. Both keys are generated on the host that is intended to be receiving the encrypted information[19], after a process called a handshake[20] is initiated by the client host.

Once the keys have been generated, the server sends the public key KU to the client as plaintext. Notice that because the key is sent as plaintext, it is possible, in fact it is likely, that key has been shared with many unintended hosts. While initially, sharing a key with a potential adversary may seem odd, but it is designed in a way such that it poses no risk to the data. Recall that I mentioned only one of the keys would be able to unlock encrypted data. The key distributed to across an internet cannot be used to decrypt. KU only possesses the capacity to preform encryption on data; however, this means that anyone that has the key can send encrypted data to the server acting as the expected client. While this is undesirable, there are methods of prevention[21].

Once the key has been sent to the client, the client encrypts their message M and sends the ciphertext back to the server. The server then decrypts the data using the private key KR, leaving them with the original plaintext M. You will notice that KR never left the server, and thus, was at no risk of being discovered by an unintentional host. This means that the server is the only host that will be able to read the message encrypted by its own key set. [9] In predicate calculus, we define functions to a public key model more closely (see 4.1). We examine an implementation before presenting the validation of the system.

For the RSA schema, we let a $pseudorandom\ generator\ G$ select a $random\ p, q \in \prod_{Prime} \mathbb{Z}$ and define n to be the product of pq. Next, we define a function $\varphi(n) \coloneqq (p-1)(q-1)$. Observe, $\lim_{n\to\infty} \varphi(n) = \infty$, meaning that as our prime integers' magnitudes increase, $\varphi(n)$ will become a larger number with more factors. Thus the construction

[19] The server.

[20] A cryptographic handshake is the process in which two hosts come to an agreement on cryptographic specifications and validate the identity of each other or a third party using *certificates*.

[21] Employment of Fundamental Principle Encryption$_1$: Messages must contain some redundancy. This redundancy helps with authentication and non-repudiation which are also important aspects of network security. For the scope of this paper we will assume that this is not a necessary precaution.

of the *hardness assumption* through our one-way function is established[22]. We must now show $\exists_e$: $\gcd(\varphi(n), e) = 1 \wedge 1 < e < \varphi(n)$. Similarly, we find $\exists_d$: $de \bmod \varphi(n) \equiv 1$. e and d will be used as the *trapdoor* allowing us to decrypt data. Now, let (KU): $[e, n]$, (KR): $[d, n]$, and plaintext message $m \in \mathcal{M}$,[23] we can define $\mathrm{Enc} := C = m^e (mod\ n)$, [24] and $\mathrm{Dec} := m = C^d\ (mod\ n)$. In other words, $\mathrm{Dec}_k(\mathrm{Enc}_k(m)) = m$.

3.2 *AES Symmetric Key Cryptosystem*

While symmetric encryption uses the traditional idea of using only one key for data manipulation, a practical implementation becomes highly complex somewhat quickly. The logic progresses as one would expect. The client initiates a handshake, the server responds and sends a key, the client encrypts the data, sends the message, and the server decrypts the message. Already, we see an issue. If we transmit the key as plaintext, anyone will be able to encrypt/decrypt data. The solution to this problem was discovered in 1976 by Diffie and Hellman. Not surprisingly, the solution was named the Diffie-Hellman Key Exchange[25]. After the key is shared, we then turn our attention to the details of the encryption process.

AES employs a 128-bit block cipher with an acceptable key size of 128-bits or 256-bits (to date. Refer to Figure 1). It was developed to replace 3DES which was extremely slow in execution which was designed to make DES withstand brute force attacks. Ultimately, the difference lies in byte wise operations compared to bit wise operations; although, a substantial amount of the algorithm is composed of *cryptographic primitives* much like DES or 3DES. For that reason, it becomes beneficial to examine DES before we look at the AES. Let's begin with a top down design before we explore the abstraction employed in the engineering.

Like any encryption algorithm, DES takes two inputs, a key (fifty-six-bits), and a piece of plaintext (in sixty-four-bit ciphers). The algorithm's encryption process can be broken into three main submodules. The plaintext is fed directly into the first submodule, T_1. The key skips over T_1, and is fed directly into the second submodule, M_{-1}. M_{-1} also receives the output of T_1 making it the only submodule to receive both the key and data to encrypt. The third submodule T^{-1} receives the output of M_{-1}, and ultimately undoes the work performed by T_1[26]. See Figure 2 for a visual representation of this process.

---

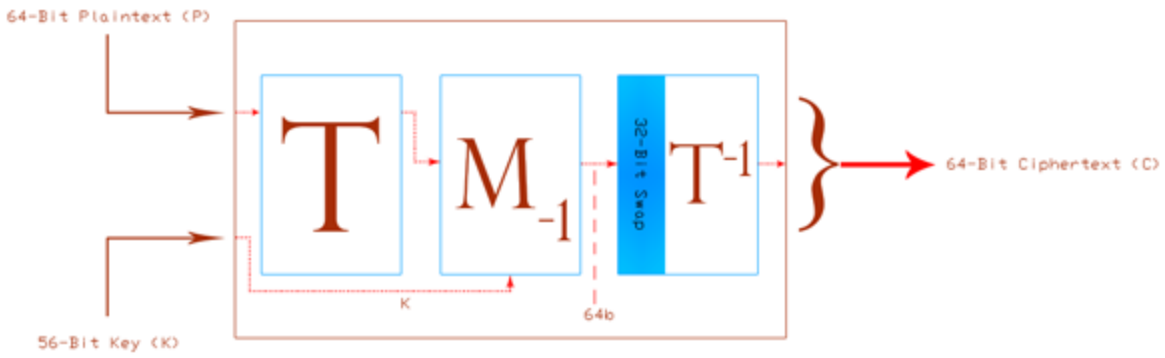
[22] Notice that in order to reverse the function, one would need to factor the large number defined by $\varphi(n)$. This is commonly known as the *factoring assumption*.

[23] $\mathcal{M}$ denotes the message space domain, ultimately, all plaintext permutations.

[24] C: Ciphertext.

[25] A solution based on the *Decisional Diffie-Hellman Assumption. By sending a series of discrete logs with certain properties to each other, they obtain a shared secret. See Diffie-Hellman Key Exchange. Further explanation is outside the scope of this paper.*

[26] You'll also notice the "32-Bit Swap" in Figure 2 on T^{-1}. This simply takes the 64-bit string of characters passed to it from M_{-1} and swaps the left 32-bits with the right 32-bits before moving on. To clarify, no bits are overwritten.



Submodules of DES

Figure 2. Shows the flow of data through the submodules of the DES Algorithm during encryption. This is extrapolated from the information from **8.2.1-The Data Encryption Standard** in *Computer Networks (Fifth Edition)* [1].

Taking a look at the first submodule, T_1, this 'T' stands for, "transposition cipher". Until now, I've mentioned them, but haven't explicitly defined their parameters. At this point, we need to add some details. The first thing we are to understand is that it uses a code word. This code word is typically predetermined or related to the key. It spans the top row of a matrix, one letter per column. This serves to number the columns according to the column's respective letter's position in the alphabet.

Password $\Longrightarrow$

W	O	R	D
4	2	3	1
P	a	s	s
w	o	r	d

$\Longrightarrow$ sdaosrPw

Data is then entered into the matrix from left to right[27], and read back out from top to bottom in the order of the column numbers.

M_{-1} is a submodule that contains sixteen sub-submodules. A different key for each sub-submodule is derived[28] from the main key and distributed to each of them. Each sub-submodule now has everything it needs and computes the *exclusive disjunction*[29] of E[30] and K_i where K_i is the key dependent upon the iteration of the sub-submodule. The result of this operation is then run through a product cipher. A product cipher is simply a chain of substitution and transposition ciphers sequentially. A substitution cipher simply allows a mapping of a code word (without duplicate letters) to the first n letters of the alphabet until the length of the code word is reached. The unused letters are filled in behind the code word alphabetically creating your ciphertext domain. Your plaintext message is then replaced by the corresponding index in the ciphertext.

[27] The data can be padded if the need be.
[28] The key generation steps are outside the scope of this paper.
[29] Otherwise known as the XOR($\oplus$) operator. The function takes an input of two parameters (true or false), and returns their *iff* one bit at a time.
[30] Expansion of the right side (32-bits) according to a fixed transposition and duplication rule.

Once flow control has moved through the sixteen sub-submodules found in M_{-1}, flow control passes to T^{-1}. T^{-1} begins with a thirty-two-bit swap[31]. Afterwards, this submodule undoes the work done by T_1, and the ciphertext is complete. The interesting thing about this algorithm is that its decryption method is simply the encryption method, in reverse.

AES can be explained best using matrices. Its implementation relies on something called the *Rijndael* function. The algorithm begins by copying the 128-bit block input into a 4x4 byte matrix. This matrix is called the *state* array, which is modified at every stage of encryption/decryption. It is also the data that gets returned after the final stage as the ciphertext. The data is inserted vertically, meaning that the first four bytes of the 128-bit plaintext occupy the entire first column. The key is also put into a square matrix before being expanded into an array of key schedule words, w. At this point, the algorithm's preliminaries have been established, and encryption can begin.

AES' encryption process consists of n number of rounds where n is dependent on the key size[32]. During rounds 1 through $(n-1)$, the same four operations[33] are permutated: *SubBytes, ShiftRows, MixColumns,* and *AddRoundKey*. It is important to note that before round one, an additional *AddRoundKey* is performed. Finally, round n performs only three operations, *SubBytes, ShiftRows,* and *AddRoundKey*. Unlike DES algorithms, AES uses an altered method for decryption. Like encryption, it begins with an iteration of *AddRoundKey*. It then moves through rounds 1 through $(n-1)$ rounds which perform the following permutation of operations: *InverseSiftRows, InverseSubBytes, AddRoundKey,* and *InverseMixColumns*. It is worth noting that the same *words* are given to opposite points of the process. In other words, $w[36, 39]$ are given to round 9 during encryption, and round 1 during decryption [9, Ch. 2, pp. 38-42].

3.3 *The Shared Secret Problem*

It is worth mentioning that neither algorithm is typically employed independently. Due to the obvious lack of complexity in the distribution of the public/private key model, it is most often selected to share the key for the symmetric model over the Diffie-Hellman Key Exchange method required to share a secret between two unassociated users. This is due to the lower cost in overall computation demanded by the AES algorithm. That key is then used for AES which then handles a majority of the computation. [1]

IV. Application of Mathematics in Semantic Encryption

4.1 *Validation of the Public/Private Key Solution*

Examining the encryption and decryption functions of the RSA again, we can see the direct application of the Discrete Log Assumption. By reducing the problem domain down to a direct application of the *hardness assumption*, we have provable security parameters. It is important to note that all cryptographic proofs imply a reduction to a one-way function, due to the status of the NP-Complete problem. This means, that in order for an adversary to *break* RSA, they would ultimately have to factor the public modulus. The best known algorithm to do this runs outside of

[31] See foot note 22.

[32] If the key size is equal to 128b, $n = 10$. If the key size is equal to 256b, $n = 14$.

[33] See [Appendix A] for a more in depth analysis of these functions.

PPT, within $O(e^{1.9(\ln(n))^{\frac{1}{3}}(\ln(\ln(n)))^{\frac{2}{3}}})$ [10, Sec. 5.6].

Extensive formal proofs exist in [10, 11]; however, at this level it proves sufficient to elaborate on the *factoring assumption*. If an adversary wants to factor n, we can start with the assertion that the smallest possible prime factor p of a *composite number* n, is $\leq \sqrt{n}$ [12, Sec. 1.2.1]. Therefore, we have that the maximum number of attempts to factor n, must be equivalent to $2^{\sqrt{n}}$. Meaning, for an appropriately sized n, there would exist 1.7976931348623 ... $x\ 10^{307}$, possible combinations. That leaves an adversary with 308 zeros before they see their first natural number when speaking in terms of their percent chance to break the cipher with a single attempt, I.E. a negligible function. There are other methods of breaking the scheme; however, the most efficient is still said to run outside of PPT. Obviously, different iterations of the solution will provide different security based on the primes provided by G; however, there are several ways to maximize the effectiveness of a generator, as mentioned in [Appendix B], [10], and [11].

4.2 *Public/Private Key Solution Vs Known Plaintext and Chosen-Plaintext*

To formally show that public key cryptography is secure in the presence of an intruder, we first define the *PPT tuple*, (Gen, Enc, Dec). Following [11, Construction 3.16], let (Gen, Enc, Dec) $= \prod$ where Gen defines, on input 1^n, choose key $k \leftarrow \{0,1\}^n$ randomly and uniformly; Enc defines, on input k and a message $m \in \{0,1\}^{\ell(n)}$, output the ciphertext c; and Dec defines, on input k and c, output the plaintext message m. Then, we say that $\prod$ has indistinguishable encryptions in the presence of an eavesdropper if $\forall_{PPT\,\mathcal{A}}\ \exists_{\eta(n)} \ni \Pr[\mathsf{PrivK}^{\mathrm{eav}}_{\mathcal{A},\prod}(n) = 1] \leq \frac{1}{2} + \eta(n)$. This probability is taken over the randomness of $\mathcal{A}$, and the key strength of the encryption process. Observe that even with knowledge of possible outcomes, e.g. $m_1\ and\ m_2$, the adversary would not be able to distinguish between the two without any probability better than fifty percent because $\lim_{n\to\infty} \eta(n)$ tends to zero[34]. See [Appendix C] for notes on any notation that is unclear.

Speaking in terms of a chosen-ciphertext attack, we follow [11, Def. 3.13], showing that $\forall_{PPT\,\mathcal{A}}\ \exists_{PPT\,\mathcal{A}'} \ni$ over many iterations, and all computable functions f and h, $\exists_{\eta(n)}$ for which the following definition holds [11, Def. 3.13]:

$$\left|\Pr\left[\mathcal{A}(1^n, \mathrm{Enc}_k(m), h(m) = f(m)\right] - \Pr\left[\mathcal{A}'\left(1^n, h(m)\right) = f(m)\right]\right| \leq\ \eta(n).$$

In this inequality, message m is chosen by the adversary and/or the adversary's algorithm, $\mathcal{A}$ or $\mathcal{A}'$ respectively, over their iterations. The probability is also effected by the randomness employed by the encryption process, key k, and the randomness of $\mathcal{A}$ and $\mathcal{A}'$. Observe what each function is given. The adversary receives a public key, an encrypted ciphertext message, a history function h, and uses all of this to guess the value of $f(m)$; whereas, there exists some algorithm $\mathcal{A}'$ that is also trying to guess the value of $f(m)$. Notice the difference however, the algorithm does not receive the encrypted message. This algorithm is said to match the adversary's chance in guessing $f(m)$ (because the inequality ultimately states, $\Pr[\mathcal{A}] - \Pr[\mathcal{A}'] \leq 0$). With the observation that each component has equal chance to guess

[34] Assuming m_1 and m_2 are of the same length.

$f(m)$, we assert that the ciphertext Enc_k reveals no information about $f(m)$, and thus a public scheme holding this property is semantically secure[35] [11, Def. 3.13].

4.3 *Validation of the AES Cryptosystem*

As I have mentioned, AES alone is not a scheme. It consists only of primitives and thus is ultimately a primitive itself. There exist heuristic proofs that it might or might not be CPA (Chosen-Plaintext Attack) secure [1]. That being said, if we examine the primitives it is made out of, we do have some provable security. Before we look at that, it is important to note that this does not make it any less secure than asymmetric encryption. Due to the fact AES takes on no one firm algebraic construction, it is hard to imagine that one could find one to reverse it without the key.

The provable security in AES stems from S-box (substitution cipher) analysis. This is said to be a *hard* problem because of the finite field inversion operation based on the Euclidean algorithm[36]. The operation ultimately is a division of polynomials until they can no longer be reduced. In doing so, we say that the *MixColumns* operation makes the distribution of the substitution appear to be uniform (random) without the key.

4.4 *AES Vs Chosen-Plaintext*

AES can in fact be modified to satisfy IND-CPA security. This is a property said to be equivalent to semantic security. This is achieved by running the cipher in a so-called mode; namely, CBC. Within the CBC mode, each plaintext block (each iteration of the algorithm) is XORed with the ciphertext of the previous block before being run through the algorithm. The first block is XORed with a randomly chosen IV (Initialization Vector). The IV then has to be sent over the network with the ciphertext, otherwise, it would be impossible to decrypt the message [1].

CBC holds the property, and thus is considered both IND-CPA secure, and KPA (Known-Plaintext Attack) secure. We show that symmetric schemes have the property by following [13]. To begin we let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ represent a symmetric key construction, similar to the way we did for public key encryption. Next we need an adversary $\mathcal{A}$ which has access to an oracle LR (meaning left **or** right). This oracles job is to accept a message pair of equal length strings from $\mathcal{A}$ and return a cipher text. This is written $C \leftarrow \mathcal{E}_k(M_b)$, where b distinguishes which message $\mathcal{E}$ has encrypted with key k. The space that $\mathcal{A}$ and an oracle *live* in is called a *world*. The world is chosen only once at the beginning.

While each world behaves in a similar way, there are some differences. We define the left world with $Game\ Left_{\mathcal{SE}}$ which holds two behaviors. The first is an initialization in which a key is initialized, $k \leftarrow \mathcal{K}$.[37] The second is called LR taking two arguments, (M_0, M_1). These are the message pair that has been passed by the adversary (similar to a remote procedure call). LR then returns $C \leftarrow \mathcal{E}_k(M_0)$. $Game\ Right_{\mathcal{SE}}$ behaves exactly the same except it returns $C \leftarrow \mathcal{E}_k(M_1)$. For an adversary to gain an advantage, after talking to an oracle for some time, he needs to be able to tell which oracle he has been interacting with. This is called the IND-CPA advantage of $\mathcal{A}$, and is defined in [13] by

[35] When it is infeasible an adversary will be able to derive meaningful knowledge about a message from its plaintext and corresponding public key.

[36] This is an extremely simplified explanation.

[37] Key space $\mathcal{K}$.

$$\mathsf{Adv}_{\mathcal{SE}}^{ind-cpa}(\mathcal{A}) \coloneqq \Pr\left[\mathsf{Right}_{\mathcal{SE}}^{\mathcal{A}} \Rightarrow 1\right] - \Pr[\mathsf{Left}_{\mathcal{SE}}^{\mathcal{A}} \Rightarrow 1].$$

Observe that if $\mathsf{Adv}_{\mathcal{SE}}^{ind-cpa}(\mathcal{A})$ is small, it means that probability of $\mathcal{A}$ existing in the right world, and the probability of $\mathcal{A}$ existing in the left world are about equal; therefore, $\mathcal{A}$ is having a hard time telling which world it's in. If either probability becomes more than one half, $\mathcal{A}$ is said to have an advantage over the encryption algorithm and it is considered not secure. It is worth noting that we assume the adversary is using a *practical* amount of resources (like oracle quires or computation time), and there is essential randomness not explicitly notated from the encryption function. There are other methods and cipher modes that imply semantic security; however, a cipher with this property is said to be IND-CPA secure [13].

V. Convergence to Quantum Mechanics and the One-Time Pad

5.1 *Hardness Assumption*

The main problem with using the one-time pad for network encryption is the requirement of the key size. It just becomes impractical for internet application; however quantum cryptography offers an elegant solution based on the assumption that it is impossible to perfectly recreate a photon. Namely, the assumption that quantum cloning, $U|\psi\rangle_A|e\rangle_B = |\psi\rangle_A|\psi\rangle_B$ [14][38] is hard [1].

5.2 *Quantum Cryptosystem (BB84)*

Before all, I should mention that quantum crypto only works on a fiber optic medium, and thus is too expensive for immediate deployment… for now. BB84 works on the principle that if light hits a filter polarized at 45º prior to its own polarization, it *randomly* jumps to a polarization perpendicular to the filter with equal probability. This is a fundamental theorem in quantum mechanics. See figure 3. Specifically, we have two sets of polarizing filters. S_1 is a vertical filter and a horizontal filter. This is called a rectilinear filter. S_2 then is a rectilinear filter rotated 45º. Light has properties such that when it passes through the first filter it is polarized in the direction of the filter's axes, and when it passes through the second, the intensity of the light equals α. More specifically, if $x = \theta_1 - \theta_2$, we have piecewise defined function in which,

$$\alpha :: \begin{cases} \cos(x) & if\ x \neg \perp \\ 0 & if\ x \perp \end{cases},$$

where θ_1 is equal to the angle of the filters in S_1, and θ_2 is equal to the angle of the filters in S_2.

The first step in the key exchange requires the server to assign two (of the four) directions of light to a bit number, and the other two directions to the other bit number. See figure 3. The server then shares these choices with the client, and transmits an encrypted message. The client then makes a guess about which bases (filter sets) to use to receive the message. He tells the server what he used and the server will then discard any bits he guessed incorrectly. They then perform a process called key amplification[39], and the secret is shared. The reason so much of this process can be performed through plaintext is because any adversary can also only guess which

[38] U is said to be the cloning operation, $|\psi\rangle_A$ is the state to be cloned, and $|e\rangle_B$ is the initial state of the copy.
[39] A process of complex manipulations to a key.

bases to use, and if he guesses differently from the client, they won't know whether their guess is correct or not due to the fundamental theorem of quantum mechanics [1].

Filter Directions in BB84

Figure 3. Shows the filters used in BB84, and a possible combination for the assignment of light to bits. This is extrapolated from the information from **8.1.4-One-Time Pad** in *Computer Networks (Fifth Edition)* [1].

5.3 *Conclusion*

We've examined how encryption has gone from being a relatively simple conjecture, to the mathematical complex that it is today. Learning how the Internet's most popular encryption works on a fundamental level, has allowed us to explore some of the proofs involved in the mathematics behind them. The applications of modern algebraic structures like groups, one-way functions, finite fields, negligible functions, discrete logarithms, and modulus operations, has become critical in the security of data as we looked closer and closer at the reasoning behind these schemes. The examination of the theoretical properties a one-way function requires gave us a look at combating an adversary that takes an active role in breaking a cipher. Finally, we saw how a so-called unbreakable cipher can be implemented for use over a network. Due to the fact that the one-time pad has proven to be unbreakable non-heuristically, and its ability to share keys at the speed of light, it lends itself more favorable to those in use today. While one day we may see quantum cryptography's implementation across the web, how long will it last, and what's to come before then?

Appendix

A) AES Details

1) Stage Definition

- ***SubBytes:*** Substitution cipher preformed on the input block, byte-by-byte. This is often implemented as reference table to cut down on computation time. It is also useful as it can exploit properties of the finite field operations on $GF(2^8)$.
- ***ShiftRows:*** A simple permutation that is performed row by row. Each row is rotated n bytes to the left where n is dependent on the row number[40]. This step disseminates the current contents around the entire block.
- ***MixColumns:*** Each column is mixed independently of the other columns. This is done with matrix multiplication where the product is given by the content of the old column, (b), with a constant coordinate vector of four numbers, f, in $GF(2^8)$. [9, Ch. 2, pp. 39-40]

[40] I.E. Row 0 is unchanged, row 1 is rotated one byte, row 2 is rotated 2 bytes. Etc.

i. $f = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$

ii. This can be represented mathematically with this generalized formula:

$$b_i = f_{i+1,1}a_0 mod(8) \oplus f_{i+1,2}a_1 mod(8) \oplus f_{i+1,3}a_2 mod(8) \oplus f_{i+1,4}a_3 mod(8)$$

Where b_i is the new element, a is the old column, and i is the row index. This is run for each column, once for each row. Notice that since we are in $GF(2^8)$, multiplication becomes a complex operation modulo $x^8 + x^4 + x^3 + x + 1$ and a_i is a polynomial.

- ***AddRoundKey:*** XORs current block with the key for the current round into the *state* array.

2) Notice that only the *AddRoundKey* receives the key. The algorithm can be viewed as alternating operations of *XOR* encryption and the dissemination of this work across the block.

3) Each stage has an easy decryption function. For the first three, it is simply the inverse. An inverse of stage four can be achieved with the notion that $A \oplus B \oplus B = A$.

B) A Word About Pseudorandom Number Generators

- **Uniform Distribution:** The distribution of bits in a sequence should be uniform; that is, the frequency of occurrence of ones and zeros should be approximately the same.
- **Independence:** No one subsequence in the sequence can be inferred from the others.
- **Further Reading:** These two points have been equivalently stated in [9, Pg. 42]. This along with about every other topic covered in this paper warrant research papers in and of themselves; however, I felt it was important I say **something** about the difference between generating random numbers, and generating random numbers for encryption and security purposes. (E.g. don't seed the system clock).

C) Predicate Calculus Notation Notes

1) I write 1^n to denote a *unary* operator, as is traditional for the scheme, and use $\{0,1\}^n$ to denote the space of all *n* length vectors composed of the base two units.

2) ℓ is said to be an expansion factor equivalent to the length of message m.

3) $\mathcal{A} := adversaries$. E.g. an algorithm.

4) $\Pr[\mathsf{PrivK}^{\mathrm{eav}}_{\mathcal{A},\prod}(n) = 1] \leq \frac{1}{2} + \eta(n)$ is read, "the probability that the private key structure $\prod$ in the presence of an eavesdropper, $\mathcal{A}$, on input n, will yield $\mathcal{A}$ with a successful distinction between an encrypted message and a random bit string is less than or equal to one half."

5) I write $C \leftarrow \mathcal{E}_k(M)$ to denote the execution of encryption function $\mathcal{E}$ on k and M, and let C hold the result returned.

6) I write $\_ \Rightarrow 1$ to mean the left hand side condition is true.

Acknowledgements

Thank you to Dr. Guercio for giving me the opportunity to research this topic.

References

[1] D. J. Wetherall and A. S. Tanenbaum, "Network Security," in *Computer Networks*, 5th ed. Boston, MA: Pearson, 2011, ch. 8, sec. 1-2, pp. 776-792.
[2] D. Hofheinz et al., "Practical Chosen Ciphertext Secure Encryption from Factoring," in *Journal of Cryptology*., J. Cryptol. 2013, 102-118.
[3] M. Jackob. (2001, August 8). *History of Encryption (2nd ed.)* [Online]. Available: http://www.sans.org/reading-room/whitepapers/vpns/history-encryption-730
[4] *Announcing the Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, 2001.
[5] *Federal Information Processing Standards Publication*, FIPS Pub 46-3, 1999.
[6] B. Schaller, "The Origin, Nature, and Implications of "Moore's Law"," unpublished.
[7] *Recommendation for Key Management*, NIST Special Publication 800-57, 2007.
[8] D. Terr. (2015, January 23). *Polynomial Time* [Online]. Available: http://mathworld.wolfram.com/PolynomialTime.html
[9] W. Stallings. *Network Security Essentials: Applications and Standards*. Upper Saddle River, NJ: Prentice, 2011.
[10] D. Stinson. *Cryptography: Theory and Practice*. Boca Raton, FL: CRC Press, 1995.
[11] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Boca Raton, FL: CRC Press, 2007.
[12] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. NY: Oxford, 1968.
[13] M. Bellare and P. Rogaway, "Symmetric Encryption," in *Introduction to Modern Cryptography*, 1st ed. San Diego, CA: UCSD, 2005, ch. 4, sec. 4.4.1, pp. 1-11.
[14] V. Bužek and M. Hillery, "Quantum Copying: Beyond the No-Cloning Theorem," Physical Review A, vol. 54, no. 3, pp. 1844-1852, Sept. 1996.