Multiple Algorithm Aperiodic Cryptosystem

Rick G. Braddy - W5FCX, 9211 West Road, Houston, TX 77064

***Abstract*—MAACS is a software architecture which layers multiple standard cipher algorithms together to increase the overall strength of the encrypted data stream, without modifying the standard algorithms, which have been heavily vetted from a cryptanalysis perspective and are already widely deployed.**

This layering technique makes the combination of algorithms significantly more resistant to traditional cryptanalysis attacks, while also addressing increased concerns over future quantum computing attacks against these algorithms. For example, concerns over published attacks against 256-bit AES and potential future attacks emanating from Grover's algorithm and other unknowns in the quantum computing era are explored. Several approaches for evolving AES are considered.

MAACS uses a concept termed "aperiodic modulation", whereby of a set of cryptographic algorithms are modified by another, independent software algorithm in an unpredictable, pseudo-random manner. In the MAACS reference architecture, a 256-bit ChaCha20 cryptographically secure pseudorandom number generator (CSPRNG) drives an aperiodic modulator that orchestrates encryption/decryption processes multiplexed across eight parallel 128-bit AES instances, yielding a cryptosystem with a composite strength of 1,376 bits or more in larger configurations.

***Index Terms*—encryption cryptography AES ChaCha20 quantum computing**

I. Introduction

Our modern world, where virtually everything is automated, operates on data. Data fuels and controls most everything now that the world has become so highly computerized. Everything from banking, web browsing, credit card transactions, online purchases, and healthcare systems to our cars, phones, networks, homes, offices and even many appliances all operate on and produce data. And most things are connected via the Internet and myriad other networks. Even modern currency like bitcoin are fundamentally just encrypted, authenticated data. The proliferation of data combined with an unprecedented level of global network connectivity has created security and privacy challenges that are unlikely to subside in a technologically advanced civilization. As a result, cryptographic technology has evolved from a largely military and commercial endeavor into one that touches most humankind today.

Every time we browse the Internet, we use encryption. Each time we buy something we rely upon encryption to keep our transaction, credit card and personal information secure. Each phone call we make is now digital instead of analog and relies upon encryption to prevent eavesdropping and invasion of our privacy. Every text we get is encrypted. Our healthcare records, financial data, personal data and most business data are now encrypted to protect against unauthorized access.

The value being protected by encryption technology has become incalculable. This fact is not lost on the bad guys in our world, who include hackers, thieves and nation-state sponsored organizations who each have their own objectives and are both skilled and highly motivated to break into this global treasure trove of data. If they can break through the veils of privacy and secrecy afforded by encryption technology, their objectives are readily achieved and the costs to society, businesses and individuals and our privacy is unthinkable.

As a result of the stakes, strong encryption technology is now widely available globally and not so restricted and coveted by the military and intelligence communities as it once was. It is clear today that encryption has become like electrical power, water and food - commodities that human beings cannot live without. Because society is so dependent upon data for everything, and strong encryption is required to protect that data, encryption has evolved into a critical ingredient to sustain our way of life.

A. How Strong is Strong Enough?

When we consider the importance of protecting our data, infrastructure and privacy, one question we must ask is how strong does the encryption need to be? And how do we know that it's strong enough?

Today's most commonly accepted encryption standard is AES [7], established by NIST in 2001 as the new encryption standard to replace the aging DES and triple DES standards. A new standard was required because the old DES standard was finally acknowledged to be insufficient to withstand attacks using modern computers, which were substantially faster and better networked than when DES was originally designed by IBM in 1972.

Most experts today seem to believe that the AES algorithm is strong enough to withstand all known attacks, provided the implementation is sound and protected against certain information leakage that can be exploited via side-channel, power analysis and boomerang attacks.

However, the reality is that related-key attacks [8] on the full AES-256 show a key recovery attack that works for all the keys and has a $2^{99.5}$ time and data complexity (instead of the expected 2^{256} for a brute force attack). Another attack [9] resulted in key-recovery with total complexity of just 2^{131} time and 2^{65} memory. These attacks show weaknesses in the AES-256 key schedule, leading its authors to recommend "new design criteria for the key schedules of the block ciphers".

These attacks showed how AES-256, which is supposed to exhibit 256-bits of strength, may be no stronger than AES-128, perhaps less. There are many opinions that express how computationally infeasible it is to brute force a 128-bit encryption algorithm, much less one with a 256-bit key. Reality is that with modern cryptanalytics, symmetric block ciphers are continually being stressed and even broken when fewer rounds are involved. The safety margins of what

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

constitutes enough rounds could continue to diminish overtime with further analytic progress, faster computers and new technologies.

Just as Enigma was broken during WW II by advances in early computing machinery, DES and 3DES were also eventually broken as computer technology advanced. According to NIST [10], DES was originally specified in FIPS 46, The Data Encryption Standard, which became effective July 1977. It was reaffirmed in 1983, 1988, 1993, and 1999. Then DES was formally withdrawn in November 2017 in NIST SP 800-67 REV. 2. Clearly industry recognized the demise of 3DES well ahead of NIST formally disapproving its continued use by the federal government. So, DES saw a total useful lifespan of less than 40 years.

AES was first introduced as a standard by NIST in 2001. If history can be used as a rule of thumb, one might expect AES to remain effective for another 40 years; i.e., until 2041. However, computer technology is advancing at a much faster pace than during last century, and the Internet now makes global scale computing jobs much more cost effective. And there are even more worrisome technology trends on the near-term horizon: quantum computing and new forms of AI. It's just a matter of time until these technologies become an attacker's best friend.

So, history teaches us that every encryption algorithm is destined to be broken, especially when the stakes are high enough to involve nation states and their tremendous resources, as we saw during WW II. It's estimated that the regular breaks of Enigma reduced the war against Germany by several years and saved millions of lives. We now know the USA successfully broke critical Japanese cryptosystems, leading to victory at Midway, that ultimately cost Japan its stranglehold on the Pacific.

History also teaches that the breaking of cryptographic systems has consequences. And it shows us that when a crypto system has been broken, adversaries go to great lengths to conceal the fact that they're reading what is supposed to be confidential, secured information. It's usually not until decades or more later that the knowledge of these disastrous cryptographic breaks become declassified or publicly known. Just because a cipher is touted as secure or thought to be certainly doesn't guarantee it is so.

Given the trillions of dollars, global stability, our privacy, and national security that are all at stake, it's clear we dare not wait until it's too late to find contemporary encryption algorithms' eventual replacements. It's important to start early, so there's enough time to develop, analyze, perfect and then plan the next replacement cycle.

Just as it was impossible to know a priori when DES and 3DES would each meet their demise, it's equally difficult to know when AES will reach the end of its useful life. But once that happens, what will we do? Perhaps we will create Triple AES, and history will repeat itself.

It's likely that billions of dollars have been sunk into AES and related infrastructure at this stage, given how widely deployed it is today. AES is even built into many CPU's, to make it much faster and more secure from prying eyes. It's a shame to have to acknowledge that we could be facing another NIST standards contest to devise an AES replacement within our lifetimes, but that appears to be more likely the case than not.

Finally, AI is another technological frontier that brings more dimensions of unknowns to today's encryption technology and its lifespan. Scientists with Google Brain project [13] are experimenting with neural networks for both encryption and decryption and conclude "neural networks may be useful not only for cryptographic protections but also for attacks" in the future.

One thing is certain. Technology will continue to advance and accelerate. What wasn't possible last century may become possible tomorrow or next year. And with the rise of quantum computers many already expect modern encryption could become obsolete, including researchers at IBM. In May of 2018, the Head of IBM Research in [14] "warns of instant breaking of encryption by quantum computers: and urges everyone to 'Move your data today', because 'this could happen in a little more than five years because of advances in quantum computer technologies'."

B. How Quantum Computers Could Change Everything

Quantum computers introduce a new capability to carry out exhaustive key searches in practical timeframes. For example, in [11] Applying Grover's algorithm to AES: quantum resource estimates examined the likely quantum computing resource requirements to exhaustively search the AES-128, AES-192 and AES-256 keyspaces. It concluded when realizing AES, only SubBytes involves T-gates, SubBytes is called a minimum of 296 times as in AES-128 and up to 420 times in AES-256. For all three AES standardized key lengths, this results in quantum circuits of quite moderate complexity. So, it seems prudent to move away from 128-bit keys when expecting the availability of at least a moderate size quantum computer.

If the AES-256 key schedule is truly as weak as has been suggested, then it's more likely that a quantum computing attack may only require an exhaustive search of the equivalent of AES-128.

Of course, the number of qubits needed to mount such attacks is estimated to be in the range of 2,953 to 6,681 range to exhaustively search AES-128 through AES-256. Today, this size quantum computer is both infeasible and prohibitively expensive. At the time of this writing, Google has announced a 72-qubit quantum computer, for example. If Moore's law were to hold for quantum computers as it did for its semiconductor cousins, then we might expect to see quantum computers in the 5,000 qubits range within 10 years - by 2028. And what if we see the advent of distributed, networked quantum computers that can be combined via the Internet to break problems up into smaller chunks and further accelerate quantum calculations? Then as IBM Research points out, we have even less time.

These possibilities do not leave us much time to chart the course for what to do next, if quantum computing truly poses the kinds of risks to modern encryption technology that we can already see just over the horizon today.

And what if someone invents a faster algorithm than Grover's for exhaustive searches that operates in polynomial time? One must believe that the advent of quantum computing will permeate computer science and lead to new breakthroughs and algorithms, once the brightest minds on the planet converge, focus and experiment with this exciting new technology.

The other area of even greater concern is, of course, the venerable RSA and its public-key compadres that rely upon the computational complexity and intensity of factoring large

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

prime numbers.

Could public key crypto already be broken today? Are the rumors of huge datacenters full of GPU's aimed at AES and RSA true? We know from history that if it's not true today, someone is almost certainly trying to make it so tomorrow.

Shor's algorithm [12], named after mathematician Peter Shor, is a quantum algorithm that runs on a quantum computer for integer factorization, formulated in 1994. Informally, it solves the following problem: Given an integer N, find its prime factors. If a quantum computer with enough qubits could operate without succumbing to quantum noise and other quantum-decoherence phenomena, then Shor's algorithm could be used to break public-key cryptography schemes, such as the widely used RSA scheme. This would reveal AES and other symmetric cipher keys, unlocking the gold mines.

Given everything that is at stake, it's clear cryptographers must start now in earnest to find answers that will protect our data, privacy and security into the next century. Efforts are underway to find new alternatives for public key exchange that are not based upon factoring large primes; however, little is being done to resolve the risk surrounding AES and our other symmetric block ciphers.

C. What can we do about AES?

When we consider the enormous investments in AES and related technologies, products and infrastructure, it's easy to see the value in finding ways to extend its useful lifespan. What options are available for doing so?

Perhaps the most obvious option is to follow the DES model, and create a 3AES that triple encrypts using the same AES algorithms. This has the advantages of being reasonably simple to do, standardize and then over time add this new 3AES layer atop of regular AES. However, if the history of DES teaches us anything, it's that this solution is little more than a short-term band aid that will not buy us a lot of time.

Another option is to increase the number of AES rounds, repair the AES-256 key schedule issue, add a new AES-512 option with 512 bits and then issue a revised AESv2 algorithm with these and other incremental improvements. This has the advantages of leveraging everything we know and trust about AES, which has been thoroughly analyzed and vetted from record levels of cryptanalysis. However, this option requires retreading the enormous AES installed base, redesigning Intel, AMD and other chips, software libraries and products, all of which have implemented the original AES algorithm according to NIST's specifications.

This brings us to a third option. Create an AESv2 that leaves AESv1 as it is today, then build a new layer on top of AESv1 that leverages the existing investments in hardware, software, products and infrastructure already incorporating it today. Make this new layer backward compatible with original AES so that software must only recompile and relink with this new AESv2 layer, while leveraging the original AES underneath as-is. And it may be possible for hardware implementations of an AESv2 to be completely transparent, only requiring a hardware upgrade to take advantage of new levels of protection. This strategy has the advantages of leveraging the existing investments in AES infrastructure and application interfaces, while addressing the ongoing risks and shortcomings of an aging AESv1.

Another related option is to replace AESv1 with an AESv2 with a stronger, faster algorithm (e.g., ChaCha20) underneath and create an AESv1 compatibility wrapper.

Whatever strategy is ultimately chosen, the one thing we already know for certain, based upon the clarity provided by history, coupled with the fuzzier images we can already see in our technology crystal balls today - AES appears to be serving us well for now, but like every encryption algorithm before it, its days are numbered. When AES' expiration date nears, we will find ourselves scrambling unless we plan for it sooner instead of reacting later, as we did when DES' time was up. Additionally, experts suggest that doubling the key lengths from 128 and 256 bits to 512 bits is another logical incremental improvement that should be enough to extend protection against quantum computing attacks (that we know about today).

So, increasing the size of the keys increases the security against brute force attacks in a post-quantum era. Increasing the rounds of symmetric block ciphers increases safety margins against known attacks. Whatever approach is ultimately chosen, increasing the maximum key length and addressing the AES-256 key schedule design are prudent at a minimum.

D. Goals

The goals of MAACS are outlined below. Whether these goals are fully attained depends upon both the successful design of MAACS and as with any encryption solution, how well implemented it is. Key goals include:

1) Build upon established, vetted encryption algorithms where possible, including AES, ChaCha20, TwoFish and other strong cryptographic algorithms
2) Create an extensible architecture that can adapt over time with evolving and new algorithms and standards to whatever level of security and protection is required
3) Address potential shortfalls of AES and other symmetric ciphers in a post-quantum computing era
4) Increase resistance to cryptanalytic attacks involving brute force key search attacks of 256-bits or more
5) Limit MAACS software-based performance overhead to no more than 20% additional CPU consumption beyond the standard encryption components it incorporates
6) Leverage at least two different standard algorithms and combine them in a manner that can resist attacks exponentially more together.

E. Assumptions

The following assumptions are being made and called out explicitly:

- Use of established encryption components that have undergone significant cryptanalysis provide the strongest foundation and best starting point for MAACS to build upon; e.g., AES and ChaCha20. There is no need to design a new symmetric encryption algorithm from scratch, as there are enough algorithms in inventory today
- The use of proven cryptographically strong pseudorandom number generators (CSPRNG) can create a strong key schedule if leveraged and implemented correctly
- Use of one cryptographically strong algorithm to pseudo randomly alter the behavior of another cryptographically strong algorithm produces a composite cryptosystem that

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

is at least as strong as the individual algorithms' bit strengths added together
- Multiplexing cryptographically strong algorithms together increases the overall security strength when done properly
- AES should be one of the reference architecture encryption algorithms because of its widespread use and acceptance as the current NIST standard
- Memory is plentiful and readily available, so the additional resource requirements posed by instantiating and operating multiple AES (or other) algorithm instances in parallel is reasonable and not too burdensome for modern computers, tablets and other portable devices
- Smaller devices such as smart cards or other devices with limited RAM or CPU do not require additional levels of security or if they do, their resources can and will be increased within the timeframe required
- Operating multiple parallel AES instances will not cause inter-instance interference of any kind as multiple instances already handle multiple sessions today
- The security benefits gained from operating multiple simultaneous algorithms together outweigh the additional costs and complexities involved because security assurance trumps added costs and complexity
- Due to published weaknesses [8] and [9] in the AES-256 key schedule, use of AES-128 may be preferred
- ChaCha20 provides the best balance between 256-bit security and performance for use as a CSPRNG and has withstood above average cryptanalysis, making it a safe bet in the MAACS reference architecture
- 256-bits is minimum size for a secure post-quantum key, and may have to increase to 512-bits within 10 years to provide enough safety margin, depending upon actual advances in quantum computing and new quantum algorithm effectiveness
- The symmetric key exchange and authentication problems posed by public-key encryption will be addressed separately, outside of MAACS, which will benefit from advances in quantum-resistant public-key or other trap-door replacement methods for secure key exchange
- Those requiring additional security assurances above and beyond AES and other standard algorithms can benefit from the additional layers of security that MAACS can deliver, without the need to develop and vet new encryption algorithms.

II. Description of MAACS

MAACS is an acronym for Multiple Algorithm Aperiodic Crypto System. MAACS is a software design based upon a simple but powerful idea: use one secure cryptographic algorithm to securely modify other secure algorithms. And use these algorithms together in a way that provides increased agility and flexibility to evolve as the preferred set of cryptographic algorithms wax and wane with time.

MAACS was born as an idea to leverage the advantages of existing ciphers using a new kind of modular, software-based cryptographic system architecture - one that's more adaptable, resilient and conducive to the need for changes than we have seen with traditional fixed symmetric block ciphers that get hard-coded into everything and then must be forklift replaced.

Its architecture leverages multiple strong encryption algorithms together to strengthen the overall cryptosystem against various forms of known and anticipated future attacks.

The basic concept of MAACS is use of a first algorithm deployed as a cryptographically secure pseudorandom number generator (CSPRNG), to modulate, or modify, how N other algorithms are used. Figure 1 shows how this process is organized.

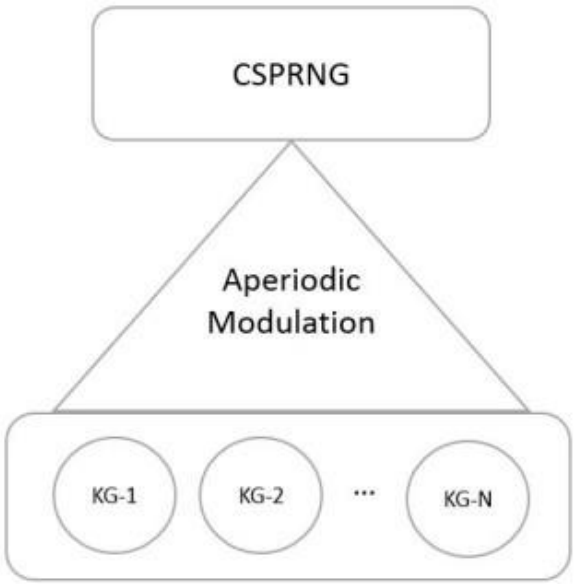


Fig. 1. MAACS Overview

In MAACS, the term key generator, or "KG", is an abbreviation that's synonymous with an encryption algorithm, such as AES, TwoFish or another cipher. Key generators are responsible for creating and processing the keys used to encipher and decipher. Modern encryption algorithms use multiple rounds of bit-blending and mathematics to obfuscate the original plaintext data, making it highly challenging for attackers to recover, while (in theory) concealing the underlying key from anything but brute force keyspace searches, which are assumed to be technically infeasible.

Aperiodic Modulation is a means of selecting and routing data across multiple encryption algorithm instances, using a pseudorandom sequencer to multiplex them together. An aperiodic sequencer determines the configuration of and order in which each KG is selected to perform its usual encrypt/decrypt process. The sequencer is directed according to a CSPRNG, which makes the sequencing cryptographically strong. Other steps further harden the combined ciphertext.

By using one hard to predict algorithm to modify the behavior of others, the resulting cipher becomes stronger, richer in complexity and much harder to attack using traditional cryptanalysis tools and methods available today.

A. *Modulation*

What is meant by modulation? It's not a term that is often heard when discussing cryptographic technology. Merriam-Webster says modulation [1] is the process of modulating a

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

carrier or signal. According to the U.S. Navy, modulation [2] is a means to systematically use an information signal (what you want to transmit) to vary some parameter of a carrier signal. For example, frequency modulation uses the information signal, $V_m(t)$ to vary the carrier frequency within some small range about its original value.

Figure 2 depicts an FM signal. Here, the carrier is at 30 Hz, and the modulating frequency is 5 Hz. The modulation index is about 3, making the peak frequency deviation about 15 Hz. That means the frequency will vary somewhere between 15 and 45 Hz. How fast the cycle is completed is a function of the modulating frequency. FM modulation is linear and predictable.

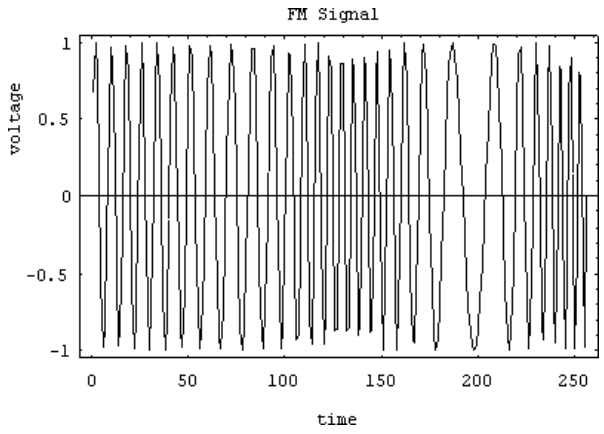


Fig. 2. A Carrier Modulated by Frequency

So, modulation causes a carrier signal to be modified by an information signal. Let's explore how one might modulate one cryptographic algorithm with another to combine their signals.

B. Aperiodic Modulation

"Aperiodic Modulation" uses an unpredictable, varying input signal to modulate a carrier signal. It's a rarely used technique that has been shown in [3] to reduce electromagnetic interference (EMI) with 10 dB suppression of the peak interference signal in switching power supplies.

For purposes of this discussion, the carrier signal we want to modulate is the data traveling through one or more encryption processes. This carrier signal flows through a set of encryption algorithms labeled KG-1 through KG-N in Figure 1. The signal used to modulate the carrier originates in the CSPRNG, which controls how Aperiodic Modulation process takes place.

This results in a modulating signal which is pseudorandom, without a recognizable pattern, being applied against one or more encryption algorithms which each produce their own pseudorandom patterns.

MAACS employs a CSPRNG created from a stream cipher to generate a pseudorandom number sequence used to: a) establish key material and initialization vectors for each of the key generator (KG) instances, and b) randomize and modulate plaintext/ciphertext routing and mixing through several KG instances. Note that for purposes of this discussion, the term "KG" refers to an instantiation of an encryption algorithm like AES, for example. The CSPRNG is keyed using a 256-bit user key, combined with a nonce, that are periodically updated throughout the life cycle of the MAACS process. This stream of pseudorandom numbers is used to build key schedules that control the aperiodic signal and modulation of the KG's.

Another perspective on Aperiodic Modulation is that it's a form of multiplexing, whereby multiple KG's are sequenced and blended together using a pseudorandom controller device.

C. MAACS Architecture

Figure 3 depicts the MAACS architecture as a block diagram with various data flows and processes. Each of the major functional areas are first discussed, followed by their interrelationships and operation.

The PERIOD Controller determines how User Keys and Nonce values are used to control the CSPRNG. As shown, CSPRNG initialization requires a CSPRNG Key and CSPRNG Nonce to begin each Period of operation. It is recommended that a random Nonce is chosen at the start of each user session and securely shared with the distant end using accepted key exchange methods to ensure session invocations are unique. External operation is not specified by MAACS, which is intended to support the same range of operating modes as AES and other block ciphers; e.g., GSM, CFB, etc.

"Periods" determine how long the CSPRNG will run with a given key schedule before being refreshed with a new schedule. Fresh pseudorandom keys and nonce values are used to seed each new Period. A Period is subdivided into "Steps." A Step is a set of encipher/decipher operations multiplexed across the array of KG's. More details about the relationship between Periods and Steps will be discussed in upcoming sections.

At the beginning of each Period, the CSPRNG generates pseudorandom numbers used to establish two key schedules, as shown in Figure 3. The Period KG Key Schedule determines how long each Period will be; i.e., how many Steps take place during the Period. It also creates the key schedule used to initialize each of the KG's, shown as KG Keys 1 through N, where N is the total number of KG's configured for use. The architecture does not limit the maximum number of KG instances; of course, practical limits always exist.

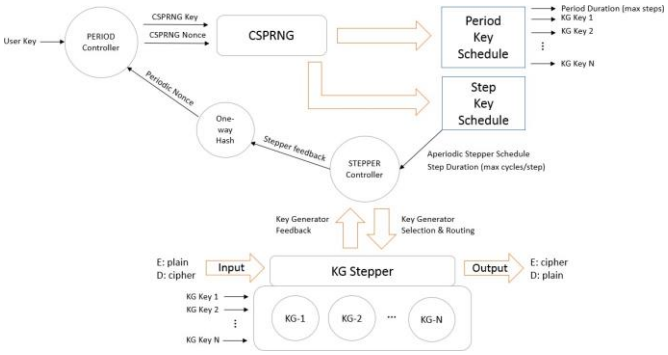


Fig. 3. MAACS Architecture

Each step is subdivided into several "Cycles", during which KG encrypt/decrypt operations take place. The Step KG Key Schedule determines how many Cycles will take place during a given Step and includes a Step Keypad.

The STEPPER Controller is the heart of the system, where Aperiodic Modulation is directed. This process directs the KG

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

Stepper function as to which KG instance to select and use during each Cycle, thereby routing the KG Input and Output flows through the selected KG. The Stepper Controller is also responsible for maintaining the Step Keypad to ensure it remains sufficiently randomized and unpredictable.

Finally, at the end of each Period, the STEPPER Controller submits its latest Step Keypad as content for the One-way Hash function, which generates a new Period Nonce, which seeds the next Period and the entire process starts over with a completely new set of key schedules, keypads and aperiodic modulation parameters.

This closed-loop, aperiodic system uses multiple layers of pseudorandom numbers, key schedules and other non-linear functions to add unpredictability. The resulting CSPRNG output, key schedules and aperiodic modulation processes vary considerably over long periods of usage.

D. Keeping Track of Aperiodic Time

In order to effect Aperiodic Modulation, one must have a system for making the system's "clock" and operational tempo both aperiodic and unpredictable. To accomplish this, a layered time model is used.

Figure 4 show the relative time relationships across Periods, Steps and Cycles. Each Period consists of a pseudorandom number of Steps, not less than a minimum number and not more than a maximum. The Period is the first aperiodic value with varying duration.

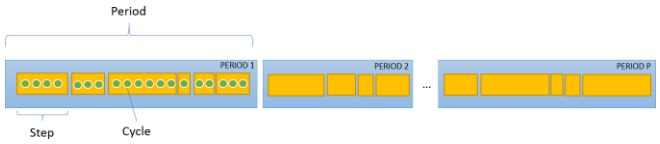


Fig. 4. Timing Diagram for Periods, Steps and Cycles

Another purpose of each Period is to limit how long the CSPRNG and KG's can operate using the same key and nonce combination. Period is expressed as a pseudo-random number of Steps; for example, a given Period could involve millions of Steps before the key schedule is rebuilt using a new Nonce value for the CSPRNG. The exact number of steps for a given Period is one of the Period Key Schedule values derived from the CSPRNG.

Each Step consists of a pseudorandom number of Cycles, as depicted by the yellow boxes in Figure 4. At the start of each Step, the CSPRNG provides a unique number of Cycles for that Step. By default, a Step consists of as few as 1,024 cycles and a maximum of 65,535 cycles. The minimum is established to limit the number of times the CSPRNG is called upon to initialize the Step variables. The maximum limits reuse of Step-specific variables. This wide duration range of a Step increases the system's variability.

Each Step operates according to the Step Keypad, a dynamic key schedule that is constantly changing throughout each Step and across the entire Period. Each Cycle proceeds according to the configuration information contained within the Step Keypad.

The KG Stepper function is driven once per Cycle by the STEPPER Controller. The KG Stepper routes the plaintext (ciphertext) through the selected KG instance, producing its ciphertext (plaintext) output; e.g., which of the AES instances are used to process a given segment of 128-bit data.

This layered approach to multiplexing across KG's using a pseudo-random stepping function creates a level of unpredictability and scale not possible using singleton ciphers. The combination of pseudo-random, Aperiodic Modulation across many KG instances is game changing and intended to confound traditional cryptanalyses, which are typically aimed at a singleton cipher algorithm today.

E. The Step Keypad

The Step Keypad is a non-recurring, one-time pad initialized by the CSPRNG key flow labeled "Aperiodic Stepper Schedule" in Figure 3. It is initially generated during the creation of each Period. To conserve memory, its default length is 4096 entries. For more secure implementations where memory is plentiful, much larger keypads could be used.

Since each keypad entry's creation requires cycling the CSPRNG, longer keypads and schedules extend the time required to reset for each new Period, so a balance between security and performance is needed; hence the default size of 4096. During Period initialization, this keypad array is populated using pseudorandom numbers from the CSPRNG.

There are Step-specific values used in conjunction with the keypad and stepper function that must be created before the onset of each Step:

1) a Step Pre-Whitening value,
2) a Step Post-Whitening value,
3) a Shift Constant value which are unique per Step,
4) a Repeat Threshold value,
5) a Discard Threshold value,
6) a Rewind Threshold value,
7) a Skip Limit, and
8) a KG Whitening key.

Note the repeat and discard threshold values must be between 1/10 the number of Cycles in the Step and the total number of Cycles in the Step.

The maximum Skip Limit value is 1/128th the size of the Step Keypad; i.e., for the default 4096 entry keypad, the maximum Skip Limit is 32. This makes the Skip limit range between 1 and 32 and changes for each Step.

A minimum Rewind Threshold is calculated as 1/256th the size of the Step Keypad and determines how many rewind bits must be observed within the Step Keypad before invoking a rewind operation; i.e. the maximum rewind threshold is the size of the keypad; i.e., no rewind operations is one of the options.

The Rewind Maximum value is also computed as the number of Cycles in the Step divided by 4095; i.e., for a full Step size of 65,535 Cycles, a maximum of 16 rewind operations are allowed during the Step.

F. Step Keypad Entry Structure

The Step Keypad provides the foundation upon which the STEPPER Controller operates. As shown in Figure 5, each Step Keypad entry contains two parts: The Step Control bits and the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

KG Offset.

The KG Offset is used to calculate a relative offset from the currently selected KG to select the next KG from within the KG array. At the onset of a Period, a "KG Pointer" is initialized to index the first KG instance. From that point onward, the KG Offset determines which KG is selected for use during a given Cycle.

A Keypad Index references the current entry for a given Cycle. By default, the controller moves incrementally from one entry to the next at the end of each Cycle.

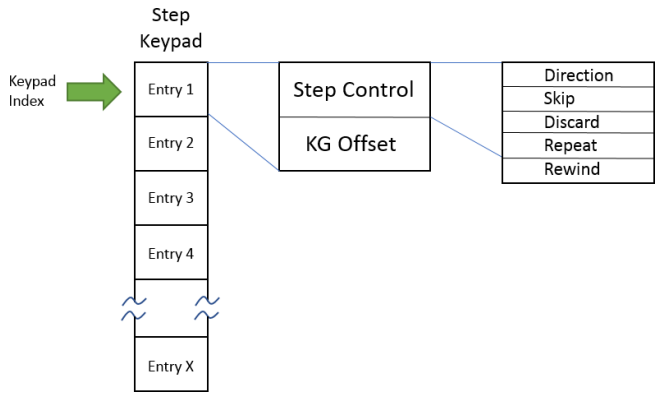


Fig. 5. Step Keypad Structure

The following Step Control bits, shown in Figure 5, are defined for each entry:

- **Direction** - a 1 indicates a positive KG Offset, a 0 equals a negative offset
- **Skip** - a 1 indicates that a skip operation is requested by this entry
- **Discard** - a 1 indicates a discard operation is requested
- **Repeat** - a 1 indicates a repeat operation is requested
- **Rewind** - a 1 indicates a rewind operation is requested.

Direction is used to choose a positive or negative KG Offset value. For example, if the KG Pointer index is currently set to 2 (with 8 total KG's configured), and the KG Offset value is 4 with a Direction value of 1, the KG Pointer will move by +4 and be set to 6; i.e., selecting the 6th KG instance to be used for this Cycle.

However, if the Direction bit had been 0 instead, with a KG Offset of 3, the offset value is -3, causing a negative wrap-around of the KG Pointer index to 7. Overflow and underflow always wrap around, so the KG Pointer index remains valid within proper bounds of the KG array.

A Skip operation causes the current entry in the keypad to be skipped (with no further processing), moving the Keypad Index ahead by one entry. The maximum number of consecutive entries that may be skipped is determined by the step's Skip Limit.

Skip operations take priority over all other operations. In some sense, the skip operation is analogous to an irregularly clocked shift register, as taught by [5] R. Rueppel in "Analysis and design of stream ciphers". Irregular clocking introduces non-linearity. E. Dawson et al in [6] "The LILI- 128 Keystream Generator" discuss making use of the good keystream properties while avoiding the inherent linear predictability of LFSR sequences, many constructions introduce nonlinearity by applying a nonlinear function to the outputs of regularly clocked LFSRs or by irregular clocking of the LFSRs.

While we are obviously not actually clocking LFSR's in this embodiment, we are shifting across the Step Keypad while simultaneously rebuilding keypad entries as we go. Each of the Step Control bits are intended to introduce non-linearity and unpredictability into the Aperiodic Modulation process. By skipping entries at pseudorandom times, we are irregularly altering what would otherwise be sequential, linear access to the Step Keypad. As we shall see, there are many non-linear functions employed.

A Discard operation causes the selected KG to proceed and run (encipher/decipher as usual), but then the results from that operation are discarded and the stepper behaves as if that operation never took place. When an entry's Discard bit is set, a Discard Counter is incremented each cycle. If the Discard Counter equals the Step's pseudorandom Discard Threshold value, then the discard operation proceeds; otherwise, only the counter is affected, and no discard takes place during this Cycle. Upon reaching the Discard Threshold, the counter is reset to zero to restart the count until next discard.

Discards are thought to be an effective countermeasure to power analysis [4], whereby externally it may appear the KG has acted, but that action did not advance the encryption data flow and is instead further masking KG multiplexing patterns.

Repeat operations cause the same KG Pointer index, and its corresponding KG, to be used over again, instead of using the keypad entry's normal Direction and KG Offset (which are ignored during a Repeat operation). When an entry's Repeat bit is set, a Repeat Counter is incremented. If the Repeat Counter equals the Step's pseudorandom Repeat Threshold value, then the repeat operation proceeds by reusing the same KG again, regardless of what this entry would normally do. If the Repeat Threshold was not reached, only the counter is affected, and no repeat operation takes place during this Cycle.

Upon reaching the Repeat Threshold, the Repeat Counter is reset to zero to restart the count until next discard. Repeat operations periodically ensure a KG gets used sequentially.

Finally, the Rewind bit can trigger the Keypad Index to become reset to its initial position, pointing at the first entry in the Step Keypad, whenever the Rewind Threshold value is reached. Rewind operations further randomize the Step Keypad's use and refresh by masking the index position into a fixed length keypad, further confounding attempts to analyze and model the STEPPER Controller. The Rewind Counter operates in a similar manner as the other threshold counters.

The Step Control bits introduce different non-linear disturbances into the behavior of the STEPPER Controller and its Step Keypad, increasing scheduling chaos and improving the overall unpredictability of the aperiodic process. This raises the bar for cryptanalysts attacking MAACS and delivers on the promise of truly "aperiodic" operation.

G. STEPPER Controller Operation

The STEPPER Controller begins at the first keypad location, then advances one element at a time until all Step Keypad entries are used, at which point the KG Stepper's pad index

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

wraps back around to the first entry (unless a skip, repeat or rewind operation is invoked, which alter how the stepping function proceeds and how keypad entries are rebuilt).

After each Step completes all its Cycles, the Stepper's pad index remains at its current location; i.e., each subsequent Step continues from the last Step's Keypad Index.

The controller is responsible for directing the KG Stepper function and for managing and refreshing the Step Keypad. The Step Keypad is intended to act similar to a secret one-time pad, except it must be dynamically updated as it gets used. This is much more efficient than using the CSPRNG to regenerate the keypad and its 4096 entries after each step, which would be prohibitively expensive. Figure 6 depicts the lightweight keypad entry refresh process, which is discussed below.

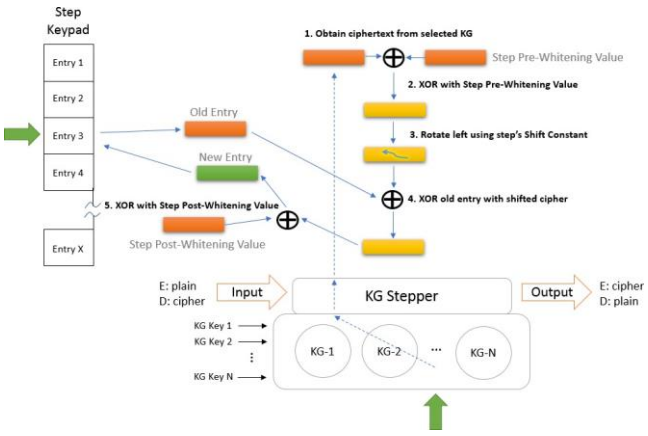


Fig. 6. Step Keypad Refresh Process

After each Cycle, ciphertext produced by the selected KG during encryption (or entering the KG during decryption) is copied from the KG via the KG Stepper and delivered back to the STEPPER Controller, where Step Keypad refresh is handled. This ciphertext temporary copy is permutated into a new keypad entry using the process depicted by Figure 6.

The ciphertext is first combined with the step's Pre-Whitening value via XOR operation, the result is rotated left by the step's Shift Constant number of bits, then combined with the old keypad entry using an XOR operation, and then finally XOR'd again with the Post-Whitening value. This permutation process produces a new pad value to replace the used one-time pad entry.

In this way, the keypad is continually refreshed using a combination of fresh cipher material derived across different KG's, each of which is operating on its own unique key during the Period. By processing the selected KG ciphertext using the Step's unique Pre-Whitening, Shift Constant and Post-Whitening values, we are ensured the same one-time keypad is indeed only used once per Step and fresh key material is generated during each Cycle.

This Step Keypad strategy is efficient, non-linear and un-predictable. It's efficient because it's fast to copy the KG's ciphertext values and process it with XOR and shift operations. It also constitutes a form of cipher feedback being introduced back into the Step Keypad and Stepper function, adding to the security of the system throughout each Period.

It's non-linear and unpredictable because it leverages the ciphertext which is produced across many pseudo-randomly multiplexed KG's, combined with the step-specific whitening and shift constants that change each Step. It's further non-linear because the stepper function sometimes skips, repeats and rewinds.

Given a reasonable upper bound of 65,535 Cycles per Step, coupled with a 4K Step Keypad that incorporates cipher feedback from an array of KG's, it's extremely unlikely the KG multiplexing pattern or the keypad will repeat before the Step reset occurs. The use of KG skip and repeat operations across multiple KG's makes it very challenging to mount power analysis against the KG's, because KG's of the same algorithm generate similar external signals, yet each is operating on its own key schedule and sub-keys. The multitude of mixed signals should confound side-channel analysis by providing a soup of confusing signals that don't correlate to a particular KG.

Even if a successful, practical attack were somehow possible to mount against the STEPPER Controller that could predict which KG's would be used and in which order (and when they are repeating vs. skipping), then there's the next layers of the onion to peel - what keys were used by each of the KG's? And how does any of this lead one back past the cryptographically strong, 256-bit random number generator that's at the heart of and blocking direct access to the underlying MAACS user key?

A deeper look into these areas for cryptanalysis will ensue in a later section, as we investigate potential attack vectors that may be used against MAACS and its components.

H. Step KG Whitening

To increase the security of the system and leverage the security bits available from the CSPRNG fully, each Step uses a unique KG Whitening key. This key is initialized using the pseudorandom number stream from the CSPRNG to create a secret key that will be used during KG encrypt/decrypt operations.

The KG Whitening key gets processed each time the STEPPER Controller cycles, including during Skip and Rewind operations. During each Cycle, each byte in the KG Whitening key gets rotated using the KG Offset and Direction bit. If Direction is 1, rotation is right; otherwise its rotation is left. The results are combined using XOR with the last set of ciphertext returned from the KG Stepper; e.g., the ciphertext entering/exiting AES. In this way, a unique KG Whitening key is always available to apply during the next KG Stepper cycle.

I. KG Stepper Operation

The KG Stepper orchestrates interactions directly with each of the KG's. It owns instantiating, initializing and operating the KG's contained within the KG Array. It also marshals data into and out of each KG as plaintext and ciphertext.

The KG Stepper is controlled by the STEPPER Controller, which initially passes it the number of KG's to instantiate, their respective keys and initialization vectors and modes of operation, along with the Input and Output data I/O. From that point forward, the KG Stepper is called upon to process the selected KG and returns the ciphertext upon completion of each processing cycle.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

The KG Stepper is also responsible for using XOR to combine the KG Whitening key with the AES ciphertext (after encryption) and against the ciphertext before decryption to complete the process. This Step Whitening ensures that even if all KG keys were somehow known or compromised, the plaintext remains unrecoverable without the KG Whitening key and exact number of STEPPER Controller cycles. This whitening process is an important element of increasing the overall strength of the combined cryptosystem by blending the CSPNRG through the Aperiodic Modulation process implemented by the STEPPER Controller and KG Stepper processes.

J. Period Reset Process

The STEPPER Controller keeps track of the number of Steps that have taken place and determines when the Period's maximum step count has been reached. At that point, the STEPPER Controller initiates a Period Reset.

The Period Reset entails calling the One-way Hash function with the Step Keypad as input. The One-way Hash creates a new Period Nonce value, which is used to seed the next Period. Finally, it calls the Period Controller, which replaces the original User Key with a new key that is generated by a final set of calls to the CSPRNG. The newly generated Period Key and Period Nonce are used to reset the CSPRNG, rebuild the key schedules and kick off a new Period. Note that the original User Keys are shredded after initially keying the CSPRING, as are the newly generated Period Key and Period Nonce values (to prevent retrieval from memory during normal operation).

Upon completion of the Period Reset, the STEPPER Controller resumes its normal operation.

K. Key Schedule Operation

As shown in Figure 3, there are two key schedules: The Period Key Schedule and the Step Key Schedule. The Period Key Schedule is created first, followed by the Step Key Schedule. The MAACS configuration (not shown) determines the number of KG's that will be used, along with other configurable parameters.

The Period Controller owns the key schedules and building them, both initially and during Period resets. It uses the CSPRNG to generate a stream of pseudorandom numbers to populate the key schedules.

The Period Key Schedule is first populated with the Period Duration, the maximum number of Steps in the Period. Next, keys and initialization vectors are created for each of KG-1 through KG-N.

Next, the Step Key Schedule is constructed to include the Step Duration value, along with initializing the Aperiodic Stepper Schedule, which is used by the STEPPER Controller to create its Step Keypad.

From that point forward, the CSPRNG is consulted at the onset of each Step to initialize the Step's variables.

L. Other Potential Features

There are many potential features that can be added to MAACS to further strengthen its abilities to confound both current and future cryptanalysts, who will presumably have more powerful computing technologies and more advanced analysis techniques at their disposal for highly sophisticated attacks. The following additional features set the bar increasingly higher, perhaps pushing practical attacks out of reach for many decades.

- **Fixed vs. Variable KG's** – the Step Keypad provides for addressing up to 255 possible KG instances. While the maximum instance figure is staggering itself, the ability to dynamically choose how many KG's will be instantiated and used adds more permutations and erects a barrier to practical attacks. The number of KG's is selected by the CSPRNG and can optionally have a Min/Max range to constrain the actual number of KG's chosen.

- **Fixed vs. Variable KG Algorithms** – by default, the KG's are assumed to be of the same (homogenous) algorithm type; i.e., all AES-256, all ChaCha20. The ability to offer a mixture of heterogenous KG algorithm types, along with each KG's bit strength level, raises more barriers against practical attacks.

- **Cipher Feedback Option** – it's possible to make cipher feedback an option to support true block cipher compatible modes. In this case, it may be necessary to add Per Cycle KG Whitening to replace the cipher feedback whitening key. Note that disabling cipher feedback is undesirable if one wants to fully protect against quantum computing attacks (assuming quantum attacks remain resistant to non-determinism as they are understood today).

- **Per Cycle KG Whitening** – it's also possible to obtain a unique per Cycle key from the CSPRNG, at the cost of additional CSPRNG operations. This may be desirable to maximize security, especially if cipher feedback were not used or desired, at the cost of some reduction in performance.

Why so much flexibility, one might ask? It adds to the aperiodic variability and overall unpredictability of the cryptosystem by increasing the number of unknowns; i.e., it's increasingly difficult to attack the system if one is unsure of exactly how it is configured, which algorithms are in use and the bit strength of each one.

As a comparison, imagine if the Enigma could have operated with up to 255 simultaneous wheels, with each wheel capable of being wired for between 2^{128} and 2^{256} combinations, where the wheel settings, wheel order, wheel type and wheel usage patterns are controlled by another set of wheels capable of 2^{256} possible settings. Add double-encryption and use of cipher feedback to ensure each encrypted message stream is unique and you'd have a formidable modern-day, software-based alternative to Enigma.

MAACS incorporates this level of security, provided one has the memory resources available to implement it.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

III. MAACS Reference Architecture

The following reference architecture is one proposed baseline implementation of MAACS, along with the key components recommended for use.

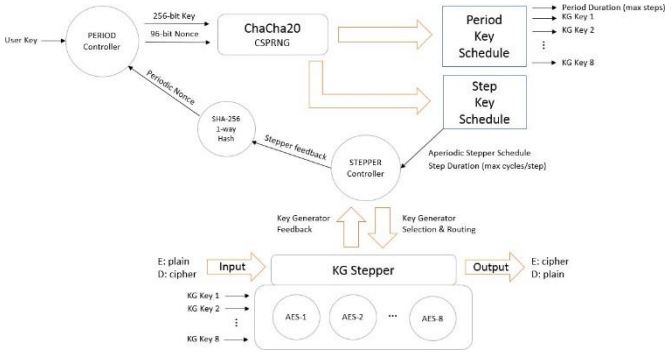


Fig. 7. MAACS Reference Architecture

As shown in Figure 7, a MAACS Reference Architecture is proposed which uses ChaCha20 as its CSPRNG, along with eight individual AES instances that serve as KG's. SHA- 256 is used as the one-way hash due to its popularity and levels of acceptance as a standard. The user may pass in an initial nonce value (not shown) that is created from a truly random external source (preferred method for more secure initialization).

Each AES KG can consist of AES in 128, 192 or 256-bit modes. As an alternate to AES, one could choose ChaCha20 with 256-bit keys. A wholly ChaCha implementation can be expected to outperform software-based AES by up to tenfold. An all ChaCha implementation might also leverage Poly1305 in lieu of SHA-256.

The PERIOD Controller is initialized by a 256-bit User Key. Future implementations can eventually accommodate larger keys, if required to stay ahead of quantum computing and other key space search threats. The Period Controller selects (or is provided by the user) a 96-bit Nonce value as a first-time initialization vector. This nonce value will be updated once per Period.

After initializing the CSPRNG, the User Key and Nonce values should be shredded to prevent future access to them.

The following default configuration settings will be used for the MAACS algorithm:

- Step Keypad Size - 4096 entries, two bytes per entry (8192 bytes total)
- Period Step Limit - 16 million Steps per Period (1 billion cycles / maximum of 65,535 cycles/step). This results in a full key reset at least once per terabyte of data encrypted/decrypted across all eight KG's combined
- Cycle Maximum Limit - 65,535 cycles per Step
- Cycle Minimum Limit - 1,024 cycles per Step
- Repeat Limit - pseudorandom number from CSPRNG between 10% and 100% of the total number of Cycles in the Step
- Discard Limit - pseudorandom number from CSPRNG between 10% and 100% of the total number of Cycles in the Step
- Skip Limit - maximum of 32 skipped entries, minimum of 8; actual skip threshold value is a pseudorandom number from CSPRNG between 1 and 32
- Rewind Limit - a maximum of 16 rewind operations is allowed during a given Step.

Every MAACS system must support at least the same set of operating modes supported by AES; e.g., CBC, OFB, CFB, CTR, whereby each such mode is implemented outside of MAACS in conjunction with AES and its usage within the MAACS system architecture. Of course, ECB mode is not recommended, and cipher feedback may be disabled to fully emulate random access block ciphers.

Other implementation details are left to the implementers of MAACS-based software embodiments.

IV. Analysis

A. Analysis of the entire cryptosystem's strength

Using the static MAACS Reference Architecture, we have combined the cryptographic strength of numerous elements, as depicted in Figure 8. The CSPRNG is keyed with 2^{352} potential settings. It generates pseudorandom sequences that drive the Aperiodic Modulation process. The Aperiodic Modulator (APM) is estimated to add 2^{16} additional permutations of the KG streams; i.e., the possible multiplexing sequences on average. An average is used because the exact sequencing is pseudo-randomly selected at runtime.

Put another way, if MAACS were an Enigma-like machine, CSPRNG represents a rotor with 2^{352} possible settings, the APM is a complex rotor with 2^{16} complexity and each KG is a rotor with up to 2^{256} settings. The total cryptographic strength, as measured in bits, is determined by:

$$CS = CSPNRG \cdot APM \cdot KG \cdot N$$

where CSPNRG strength is 256 key bits plus 96 bits of pseudorandom nonce, APM is 16-bits, N is the number of KG's, KG has a strength between 128 to 256 bits, yielding total cryptographic strength CS between 1,376 to 2,400 bits.

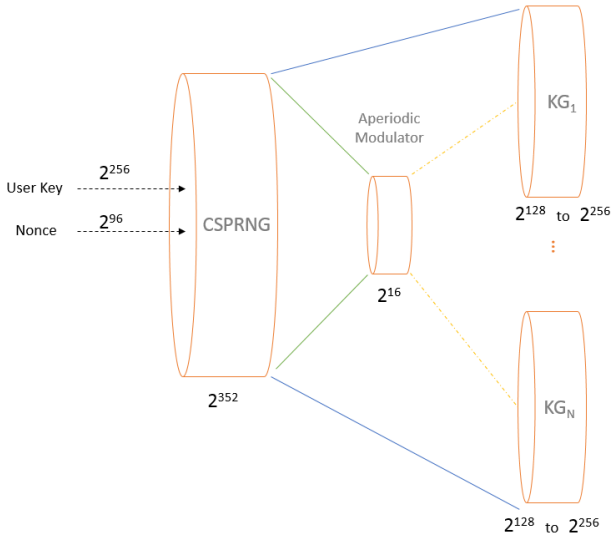


Fig. 8. Key Strengths

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

This calculation is predicated upon the following assertions:

1) A brute force attack of at least 2^{352} is required to break the CSPRNG (256-bit key and 96-bit nonce)

2) Each KG can withstand a brute force attack of between 2^{128} to 2^{256}

3) The entire set of multiplexed KG's must be individually broken to recover any usable message component due to the sequencing, whitening and cipher feedback loops

4) The Aperiodic Modulator provides randomization of KG sequencing and KG Whitening, such that the actual KG sequence cannot be recovered short of a successful brute force attack on the CSPNRG in order to recover the key schedules and Step variables involved. It's difficult to imagine how one could mount such an attack, given there are no "plain" texts to recover or compare against to measure success

5) Because cipher feedback is used within the Aperiodic Modulation process, even a brute force attack against CSPRNG is insufficient.

This analysis is based upon a static (fixed KG dimensions) configuration and does not include dynamic features like mixing different KG algorithms or variable quantities of KG instances, which add to the combinations and analysis complexity, making algebraic attacks impractical in size.

To delve deeper, let's start with the set of multiplexed KG's. Breaking each KG algorithm requires 128 to 256 bits of brute force and/or attacking multiple, simultaneous targets. There are eight distinct KG's (in the reference architecture), each with its own unique pseudorandom key and IV derived from the CSPRNG. If all eight must be successfully cracked to yield an entire message, this is 1024 to 2048 bits of combined strength (more considering pseudo-random IV).

In order to recover the message, the exact sequence of the KG's used to encrypt the message must be known, along with the KG Whitening keys. Even if we somehow knew the keys of all eight KG's, we would still have an unencrypted message due to the KG Whitening and cipher feedback processes.

Because of the sequencing, cipher feedback has a further confounding effect on attacking the system because of how non-linear the feedback is when spread across many KG's and mixed with the KG Whitening. This is further exacerbated by Skip operations, whereby the KG cycles but the data stream does not proceed, thereby skewing the cipher feedback and whitening stream unpredictably.

So, the combination of KG Whitening, coupled with KG multiplexing and KG skipping in an unknown and unpredictable order add to the strength of the combined CSPRNG and Aperiodic Modulator processes (at 368 bits) to the KG's aggregate 1024 to 2048 bits to yield 1,376 to 2,400 bits total cryptographic strength.

While the APM and multiplexing do not add significantly to the overall cryptosystem's bit strength, the KG Whitening and KG sequencing effectively multiply the CSPRNG and KG array strengths through APM cipher mixing.

B. Analysis of the ChaCha20 CSPRNG and Key Schedule

ChaCha20 [15] was chosen as the cryptographically strong pseudorandom number generator (CSPRNG). It was chosen due to its excellent performance and abilities to withstand numerous sophisticated cryptanalytic attacks over time. It was also chosen due to its increasing industry acceptance, its use as the random number generator for Linux and IETF [16] standardization.

Differential cryptanalysis [17] concluded that a total of 12 rounds would be enough to achieve 256-bit security. The 20 rounds of ChaCha20 provides an additional 8 rounds of safety margin.

KDDI Research evaluated the security of ChaCha against known cryptanalyses and concluded in [18] there are no efficient differential analysis, linear cryptanalysis and distinguishing attack, guess and determine analysis, algebraic attack, and no successful attacks on initialization of ChaCha.

Time-memory-data tradeoff attack theoretically applies to ChaCha. The time complexity is $2^{234:67}$ using 2^{117} keystreams and 2^{176} for the IETF version of ChaCha with 32-bit counter and 96-bit IV. However, there are no time-memory-data tradeoff attacks with time complexity less than 2^{256} (brute force) by limiting the number of keystream to 2^{96}, which is a practical assumption.

The MAACS key schedules are simplistic, in that they are created using the pseudorandom number sequences produced by the CSPRNG. Since ChaCha20 has been shown to be secure, and we are using a 256-bit user key coupled with a 96-bit randomly chosen nonce, the resultant key schedules are secure up to 2^{356}.

Note that use of ChaCha20 presumes initialization with a random nonce value with enough entropy (a topic that has already been explored in a large body of existing work).

C. Analysis of the AES KG

AES is the current NIST data encryption standard. It has undergone extensive cryptanalysis and stands on its own merits. There is plenty of available research available on the strength of the AES algorithm that will not be repeated here.

However, one area of concern is the AES-256 key schedule. Related-key attacks [8] on the full AES-256 show a key recovery attack that works for all the keys and has just a $2^{99.5}$ time and data complexity (instead of 2^{256} for a brute force attack). Another attack [9] resulted in key-recovery with total complexity of just 2^{131} time and 2^{65} memory.

Assuming quantum computing can reduce these key recovery times in half, we could face an AES-256 susceptible to attacks that work for all keys in just 2^{50} time! This would be a catastrophic failure. AES-128 has not shown these weaknesses.

Additionally, AES-256 entails 14 rounds per operation. AES-128 requires just 10 rounds, a 28.5% savings in round overhead.

For these reasons, AES-128 is recommended for the baseline MACS reference architecture due to its superior overall security with the reduction in round CPU costs. The round savings should offset much, if not all, of the additional overhead posed by MAACS and its various processes (actual benchmarks are needed to confirm this assumption).

The multiplexing of independent AES-128 instances to serve as KG's results in additive strength when combined as a system. In the reference architecture, each AES-128 KG adds 128 bits of unique security strength, totaling 2^{1024} combined strength for an eight element KG array. That is, in order to recover a message, all eight AES-128 instances must collectively be breached. There

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

is no known cryptanalytic method that can be used to successfully mount anything remotely close to such an attack today.

D. Analysis of the Aperiodic Modulator

The key differentiation between the traditional cryptographic algorithms used by MAACS is how they are configured to operate together via aperiodic modulation.

A foundational assertion is that the CSPRNG is secure, as established above. All pseudorandom values derived from the CSPRNG are therefore considered secure, given the CSPRNG key and nonce remain secret.

Aperiodic Modulation involves sequencing the KG's in a pseudo-random order. If this order cannot be determined, it's impossible to recover the original message. Cipher feedback is used from the KG's during operation. This feedback emanates from a set of KG's which are sequenced in an unknown order. Successful recovery of the message relies upon the same state being reproduced as a by-product of all Aperiodic Modulation components operating in concert across the array of KG's.

Many different features complicate cryptanalysis of the Aperiodic Modulation processes. First, the number of Steps during each Period is unknown. It can vary from 1024 steps to a maximum of 65,535 steps.

Second, the Step Keypad consists of 4096 entries. This creates 2^{12} combinations per 4096 cycles. Assuming an average number of Cycles of 16384, we have 2^{14} operations per Step.

So, on average we can expect between 2^{10} and 2^{16} cycles to occur during a given Step and it is not possible to predict how many actual cycles will occur before the Step Variables and KG Whitening keys will be replaced with new ones from CSPRNG.

The STEPPER Controller operates according to the configuration information contained in the Step Keypad and Step Variables. It's responsible for directing the KG sequencing operations. If we assume the Step Keypad remains pseudorandom, then there are at most 2^{16} static operations before these variables are changed again. It should be readily possible to brute force this level of complexity. However, the STEPPER Controller function is not static.

The STEPPER Controller receives cipher feedback from an array of (eight) KG's. Each of these KG's produces ciphertext based upon its individual AES encryption process. This ciphertext is fed back into the Step Keypad randomization process, where it alters how the STEPPER Controller operates.

Nevertheless, let's assume we can somehow brute force an array of eight AES-128 algorithms through all 2^{16} possible sequencing combinations. We still have major barriers to overcome.

First, to recover the message, we must correctly recover 1024 bits of key material for all AES-128 KG's. It's already been established that the key schedules are secure due to the CSPRNG being secure. Therefore, the ability to put the STEPPER Controller through all possible combinations of sequencing yields nothing useful.

Second, the KG Whitening process is an additional step of encryption that must also be reversed. At the onset of each Step, the CSPRNG seeds the STEPPER Controller with a fresh set of KG Whitening Keys. These keys are used as an additional layer of encryption on top of AES using XOR and rotate operations.

Then, this composite ciphertext is fed back into the STEPPER Controller, where it is further used to permute the Step Keypad. The original ciphertext emanating from AES during encryption is only visible in the clear to the KG Stepper and STEPPER Controller processes. Once it has been used during whitening, it is shredded. The final composite ciphertext that exits the MAACS layer is the AES ciphertext further encrypted with the KG Whitening key.

Without the KG Whitening key and original AES ciphertexts, it's impossible to decrypt the message. And we have already established that the array of KG's requires a 2^{1024} brute force attack, which isn't feasible to anyone's knowledge.

Finally, the KG Whitening Keys are produced by the CSPRNG, which we know is secure. The ciphertext produced by the AES KG's is further secured (and obscured) by KG Whitening Keys and the corresponding XOR operation.

Therefore, we conclude there is no known, computationally feasible attack to break the Aperiodic Modulator when combined with the KG array, even if the Aperiodic Modulator is broken.

F. Analysis of potential impacts from future quantum computing

Quantum computing brings unknowns that create many concerns among today's technology and cryptographic communities. The most pressing concerns involve [20] Shor's algorithm coupled with a quantum computer's ability to factor large prime numbers in polynomial time. This threatens the very fabric of today's public-key infrastructure, which is used to authenticate secure endpoints and shuttle secret keys required for two or more parties to communicate using symmetric ciphers like AES.

Another worrisome area involves quantum computers and Grover's algorithm [19]. This is the area which most concerns MAACS and AES. In [21], "Applying Grover's algorithm to AES: quantum resource estimates", the conclusion is "it seems prudent to move away from 128-bit keys when expecting the availability of at least a moderate size quantum computer."

	#gates		depth		#qubits
k	T	Clifford	T	overall	
128	$1.19 \cdot 2^{86}$	$1.55 \cdot 2^{86}$	$1.06 \cdot 2^{80}$	$1.16 \cdot 2^{81}$	2,953
192	$1.81 \cdot 2^{118}$	$1.17 \cdot 2^{119}$	$1.21 \cdot 2^{112}$	$1.33 \cdot 2^{113}$	4,449
256	$1.41 \cdot 2^{151}$	$1.83 \cdot 2^{151}$	$1.44 \cdot 2^{144}$	$1.57 \cdot 2^{145}$	6,681

Fig. 9. Quantum resource estimates for Grover's algorithm to attack AES-k, where k := 2{128; 192; 256} [21]

As shown in Figure 9, a quantum computer will require 2,953 qubits to exhaustively search the key space of AES-128. It remains to be seen how long we shall have until quantum computers of this size can be built or networked together. The good news short-term is, quantum computing is presently expensive, so mounting practical attacks using them will likely be limited to nation states and well-funded organizations for some time.

Presumably similar concerns exist for ChaCha20, although no specific quantum computing research was found to be available.

For a time, the MAACS 256-bit key plus 96-bit random nonce should suffice; however, once quantum computing is proven to successfully scale Grover's algorithm to several hundred qubits, it will be time to be moving up to 512-bit and 1024-bit keys to extend the useful lifetime of our symmetric encryption systems.

Symmetric block ciphers appear more conducive to being modeled by a function in order to apply Grover's algorithm,

 doi:10.20944/preprints201910.0085.v1

given their relative simplicity and uniformity. One of the advantages of the complexity of MAACS and its Aperiodic Modulation process is it will be much more challenging to simplify into a suitable function that can be leveraged by Grover's algorithm operating with the limits of quantum computers.

V. Conclusions

This paper is largely exploratory, discussing ideas and concepts for increasing the strength of existing and future software-based cryptosystems. The approach leverages off-the-shelf, proven cryptographic software algorithms, and introduces a new Aperiodic Modulation apparatus that binds multiple algorithms together. This creates a cryptosystem that can withstand more extreme levels of attack, including those resulting from future quantum and supercomputers.

We have shown various methods to combine multiple encryption algorithms to increase the overall strength of a software cryptosystem. We employed a CSPRNG to seed and feed a novel Aperiodic Modulator to multiplex and sequence an array of AES instances. We proposed a reference architecture comprised of ChaCha20 and AES as two primary algorithms.

We combined these with SHA-256 as the one-way hash for securely reseeding with a new nonce at least once per terabyte of data flow. We showed how other common cipher and hashing algorithms could just as easily have been chosen instead and configured as homogenous (same algorithm) or heterogenous KG arrays.

We created a closed-loop feedback system that maintains the system's long-term security by limiting the duration that keys and circulating permutations are required to remain unique.

We relied heavily on the well-researched aspects of the popular ChaCha, AES and SHA-256 algorithms and their vetted cryptanalysis as core elements of a secure system.

We demonstrated how MAACS can extend the strength of AES from a maximum of 256-bits into a hybrid cryptosystem with up to 2,400 bits of strength with 8 KG's, which is beyond the reach of quantum computing as we understand it today. We also hypothesized about a KG array with up to 255 KG's, including a heterogenous mix of strong encryption algorithms, that can yield a galactic 65,536 bits of strength.

VI. Future Work

Independent cryptanalysis of MAACS by experts in mathematics and cryptanalytics fields, along with peer reviews, are needed to identify weaknesses, flaws or vulnerabilities that and address them before implementing.

Prior to software implementation and release to the public, an object-oriented design document addressing implementation goals and objectives is also needed.

A cross-platform reference software implementation of MAACS is needed to explore various characteristics and performance. Perhaps leveraging the CRYPTO++ library for portable ChaCha20, AES, SHA-256 and other algorithms would be a logical approach.

A test framework for better understanding and perfecting various aspects of the Aperiodic Modulator, its data structures, cipher feedback and whitening processes would be helpful.

New algorithms and strategies for Aperiodic Modulation could improve upon or replace the original design.

VII. References

1. "Definition of MODULATION." Accessed December 18, 2018. https://www.merriam-webster.com/dictionary/modulation.
2. "Frequency Modulation." Accessed December 18, 2018. https://fas.org/man/dod-101/navy/docs/es310/FM.htm.
3. Hasan, S. U., and G. E. Town. "An Aperiodic Modulation Method to Mitigate Electromagnetic Interference in Impedance Source DC–DC Converters." IEEE Transactions on Power Electronics 33, no. 9 (September 2018): 7601–8. https://doi.org/10.1109/TPEL.2017.2772909.
4. Kocher, Paul C. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," n.d., 10.

[5] R. Rueppel. Analysis and design of stream ciphers. Springer–Verlag, Berlin, 1986

[6] Dawson, E, A Clark, J Golic, W Millan, L Simpson, and L Penna. "The LILI-128 Keystream Generator," n.d., 14.

[7] "Advanced Encryption Standard." Wikipedia, December 8, 2018. https://en.wikipedia.org/w/index.php?title=Advanced_EncryptionStandard&oldid=872719186.

[8] Biryukov, Alex, and Dmitry Khovratovich. "Related-Key Cryptanalysis of the Full AES-192 and AES-256," 2009. http://eprint.iacr.org/2009/317.

[9] "(3) (PDF) Distinguisher and Related-Key Attack on the Full AES-256." ResearchGate. Accessed December 20, 2018. https://www.researchgate.net/publication/221354789_Distinguisher_and_Related-Key_Attack_on_the_Full_AES-256.

[10] Barker, Elaine, and Nicky Mouha. "Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher." Gaithersburg, MD: National Institute of Standards and Technology, November 17, 2017. https://doi.org/10.6028/NIST.SP.800-67r2.

 doi:10.20944/preprints201910.0085.v1

[11] Grassl, Markus, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. "Applying Grover's Algorithm to AES: Quantum Resource Estimates." ArXiv:1512.04965 [Quant-Ph], December 15, 2015. http://arxiv.org/abs/1512.049 65.

[12] "Shor's Algorithm." Wikipedia, December 5, 2018. https://en.wikipedia.org/w/index.php?title=Shor%27s_algorithm &oldid=872138610.

[13] Abadi, Martín, and David G. Andersen. "Learning to Protect Communications with Adversarial Neural Cryptography." ArXiv:1610.06918 [Cs], October 21, 2016. http://arxiv. org/abs/1610.06918.

[14] Foremski, Tom. "IBM Warns of Instant Breaking of Encryption by Quantum Computers: 'Move Your Data Today.'" ZDNet. Accessed December 20, 2018. https://www.zdnet.com/article/ibm-warns-of-instant-breaking-of-encryption-by-quantum-computers-move-your-data-today/.

[15] "ChaCha20 DRNG." Accessed December 21, 2018. http://www.chronox.de/chacha20_drng.html.

[16] Langley, Adam, and Yoav Nir. "ChaCha20 and Poly1305 for IETF Protocols." Accessed December 21, 2018. https://tools.ietf.org/html/rfc8439.

[17] Choudhuri, Arka Rai, and Subhamoy Maitra. "Differential Cryptanalysis of Salsa and ChaCha -- An Evaluation with a Hybrid Model," 2016. http://eprint.iacr.org/2016/377.

[18] "Security Analysis of ChaCha20-Poly1305 AEAD," n.d., 38.

[19] "Grover's Algorithm." *Wikipedia*, November 26, 2018. https://en.wikipedia.org/w/index.php?title=Grover%27s_algorithm&oldid=870715968.

[20] "Shor's Algorithm." *Wikipedia*, December 5, 2018. https://en.wikipedia.org/w/index.php?title=Shor%27s_algorithm&oldid=872138610.

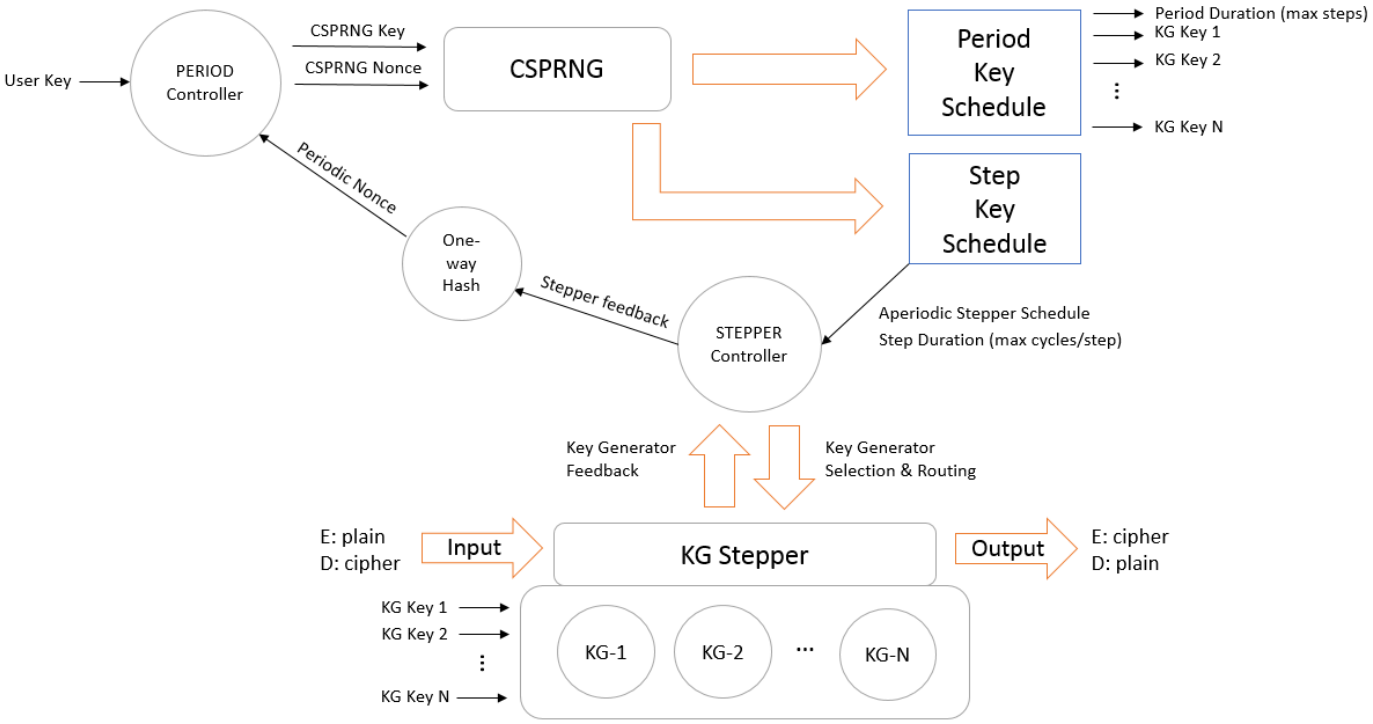
[21] Grassl, Markus, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. "Applying Grover's Algorithm to AES: Quantum Resource Estimates." *ArXiv:1512.04965 [Quant-Ph]*, December 15, 2015. http://arxiv.org/abs/1512.04965.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1

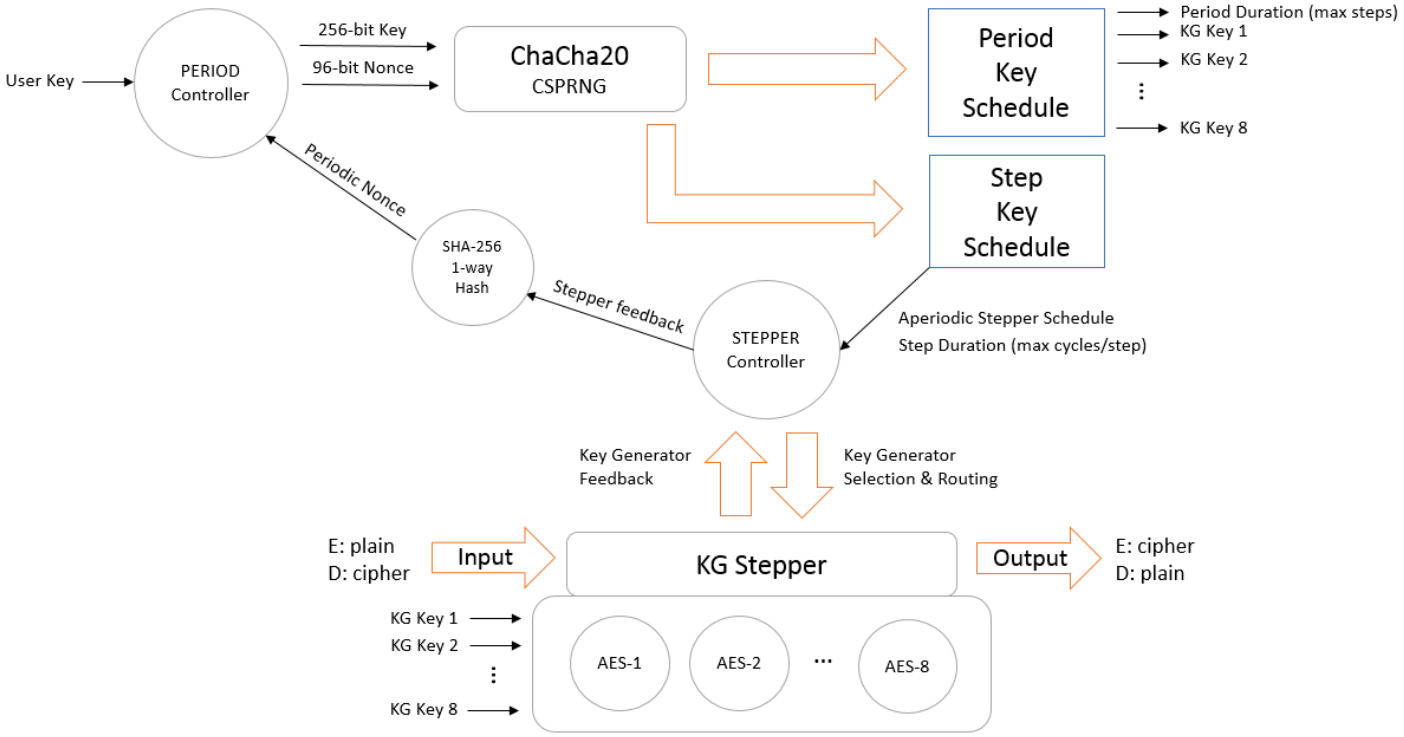
Rick G. Braddy is an innovator, leader and visionary with more than 40 years of technology experience. Rick is a serial entrepreneur, software developer and former Chief Technology Officer of CITRIX Systems and former Group Architect with BMC Software. Rick is a United States Air Force veteran, with experience in military cryptographic voice and data systems at NORAD's Cheyenne Mountain Complex from the 1970's. He currently serves as Co-founder, CTO of SoftNAS, Inc., a leading NAS and cloud data platform company.

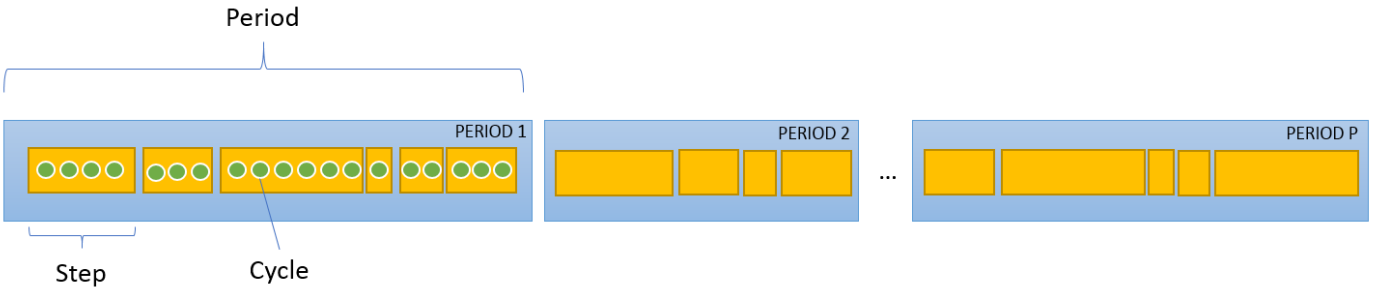
VIII. APPENDIX

A. Full-scale drawings

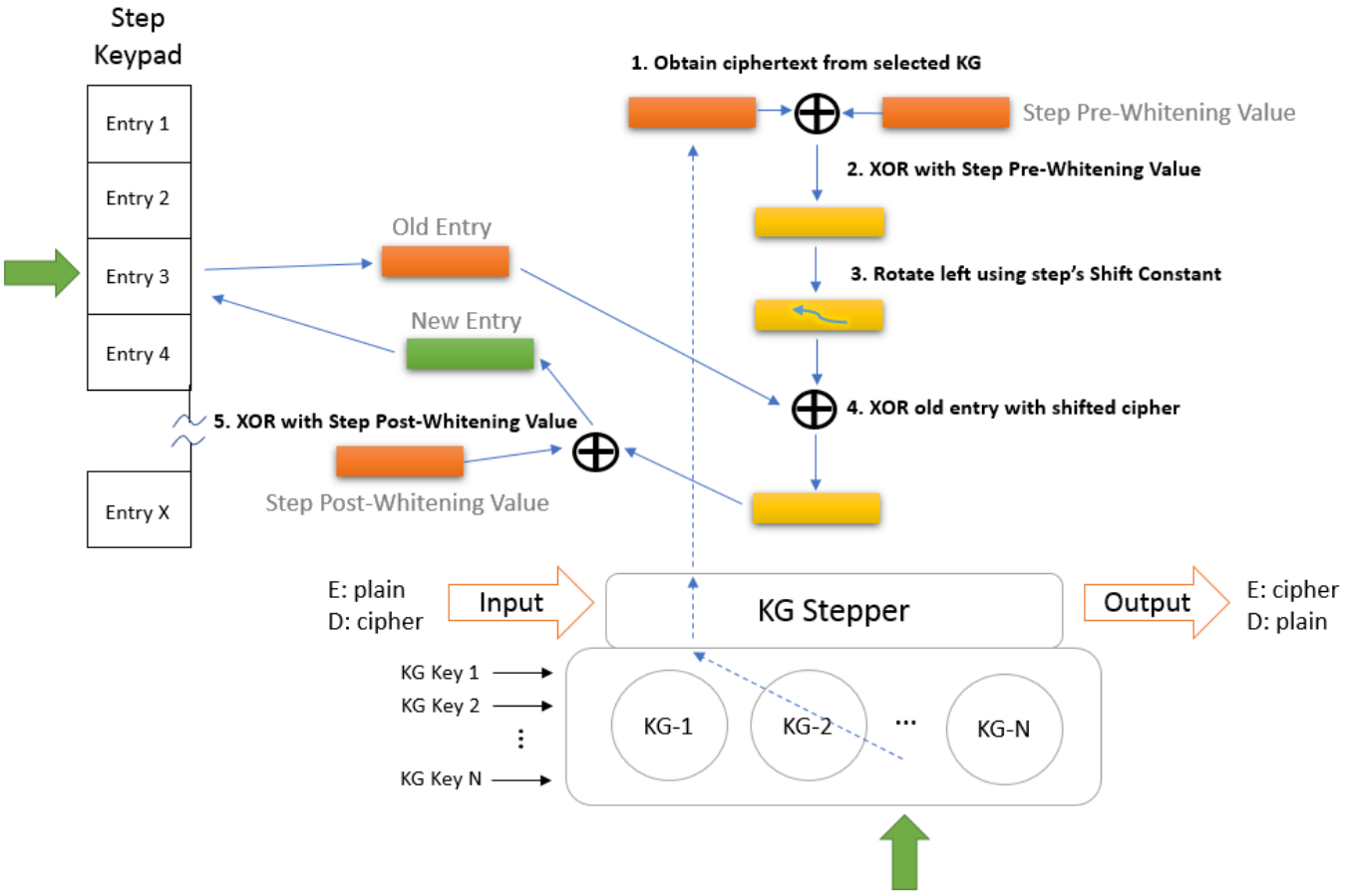


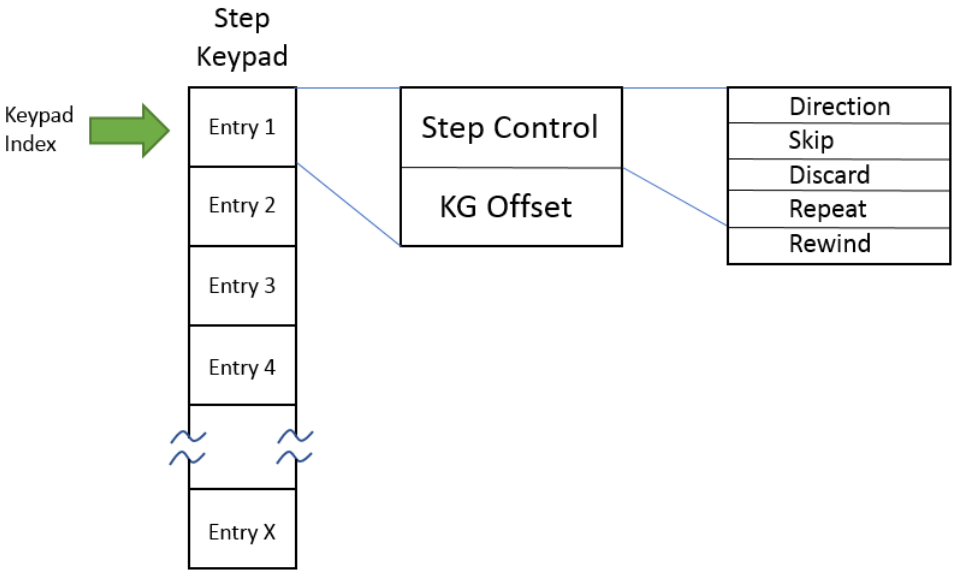
Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1





Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1





Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 8 October 2019 doi:10.20944/preprints201910.0085.v1