IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 13, 2019, accepted July 4, 2019, date of publication July 15, 2019, date of current version August 2, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2928768

Further Analysis of PRNG-Based Key Derivation Functions

JASON M. MCGINTHY, (Student Member, IEEE), AND ALAN J. MICHAELS, (Senior Member, IEEE)
Hume Center for National Security and Technology, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA
Corresponding author: Jason M. McGinthy (mjason@vt.edu)

ABSTRACT The Internet of Things (IoT) is growing at a rapid pace. With everyday applications and services becoming wirelessly networked, security still is a major concern. Many of these sensors and devices have limitations, such as low power consumption, reduced memory storage, and reduced fixed point processing capabilities. Therefore, it is imperative that high-performance security primitives are used to maximize the lifetime of these devices while minimally impacting memory storage and timing requirements. Previous work presented a residue number system (RNS)-based pseudorandom number generator (PRNG)-based key derivation function (KDF) (PKDF) that showed good initial energy-efficient performance for the IoT devices. This paper provides additional analysis on the PRNG-based security and draws a comparison to a current industry-standard KDF. Subsequently, embedded software implementations were performed on an MSP430 and MSP432 and compared with the transport layer security (TLS) 1.3 hash-based message authentication code (HMAC) key derivation function (HKDF); these results demonstrate substantial computational savings for the PKDF approach, while both pass the NIST randomness quality tests. Finally, hardware translation for the PKDF is evaluated through the Mathworks' HDL Coder toolchain and mapping for throughput and die area approximation on an Intel® Arria 10 FPGA.

INDEX TERMS Internet of Things, key derivation function, key management, lightweight, security.

I. INTRODUCTION

The Internet of Things (IoT) is growing rapidly and expanding into many areas that recently did not have Internet capabilities such as manufacturing, infrastructures, vehicles, aircraft, and healthcare. These new Internet-connected devices are being developed to reduce costs, increase safety, collect environmental data, and generally improve people's lives. Key management is a major concern in IoT due to the vast number of devices and the lack of resources available to these devices to run current protocols. One main aspect of key management is key generation for use in other security protocols such as cryptographic functions. IoT complicates key generation because keys may need to be refreshed frequently, causing more energy consumption and limiting the lifetime of the device. Therefore, an energy efficient key derivation function (KDF) must be utilized to reduce this consumption. The work presented in this paper expands upon a previous pseudorandom number generator (PRNG)-based key derivation function (PKDF) developed in [1]. The PKDF was designed to meet the needs of resource-constrained devices that will dominate the IoT market.

In order to share sensitive data between two parties, a cryptographic process can be performed to encrypt the message. The receiver then must decrypt the transmitted ciphertext to reveal the original information. Generally, there are two types of encryption schemes: asymmetric and symmetric. Asymmetric encryption typically requires longer keys, multiple transmissions to acquire the public key, and overall more computationally complex operations. Compared to symmetric functions, these shortfalls can limit their usefulness on IoT devices. On the other hand, symmetric encryption provides the same security level with much shorter keys lengths (e.g. 256 bits vs. 15,360 bits) [2].

The initial work done in [1] initially showed that keys produced from the PKDF met thresholds recommended by the NIST statistical test suite for random and pseudorandom number generators [3], but there were still remaining concerns on the quality of the key stream outputs. Therefore, further methods of validation are examined in this paper. First, the law of the iterated logarithm (LIL) testing [4] is used to increase the confidence of the pseudorandom nature

The associate editor coordinating the review of this manuscript and approving it for publication was Macarena Espinilla.

of the PKDF by examining fluctuations in the variance of output streams. Next, more entropy tests are performed to validate that no information is leaked from one key to another, providing a key element for forward secrecy. Finally, since there were no true side-by-side performance comparisons of the PKDF with any other KDF, software implementations of the PKDF and the hash-based message authentication code (HMAC) key derivation function (HKDF) [5] on two platforms: MSP430FR5994 [6] and MSP432P401R [7]. This additional analysis supports the overall goal: establish low-power security that can improve existing size, weight, and power (SWaP)-constrained IoT solutions. This is achieved by dynamically generating session keys that are used and updated quickly, limiting the value to an attacker if efforts are made to crack the individual session keys. The rest of the paper is organized as follows: Section II discusses related work in KDFs. PRNG-based strength security is examined in Section III since PRNGs are the core function of the overall PKDF. Section IV provides energy consumption results of the PKDF implemented on a low power, IoT-like device and also presents initial performance results of the PKDF hardware implementation. Finally, Section V presents the conclusions and integration of the PKDF with future work.

II. BACKGROUND

As previously mentioned, key management is a critical component of security implementations. Many cryptographic protocols rely on the use of randomly or pseudorandomly generated keys to perform encryption, decryption, and authentication functions. Cryptographic keying material must be carefully derived as to not indicate any weakness in generation or other vulnerabilities. If these procedures are weak, then an attacker can replicate the process and have knowledge of the keys. This renders all cryptographic functions useless if the key is known, exposing all previously encrypted information to unauthorized observers.

A. PKDF

The previous work in [1] focused on the design of an efficient KDF that is PRNG-agnostic. This PKDF approach allows for flexibility in the design based on the capabilities of the devices in use. The driving factor for this design was to reduce the overly-complicated functions that are not practical in an IoT device. The PKDF requires a pre-shared master key (MK) between two devices of sufficient length, n (where $n \geq 1024$ bits). The overall premise of the PKDF is that through the use of the PRNG, bits from the MK are selected pseudorandomly until the desired output session key (SK) length, k, is met. This is internally performed by reducing the PRNG output through a modulo operation by a design parameter, w, and then adding the previously reduced PRNG value to create a random walk process. This window parameter, w, bounds the random walk step that each successive PRNG value can progress through the MK. After this summation, the value is then reduced modulo by the length of the MK, n, to ensure that the bit address stays within the bounds of

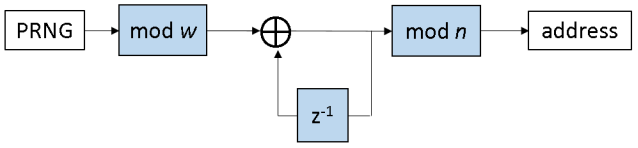

FIGURE 1. Diagram of the PKDF developed in previous work [1].

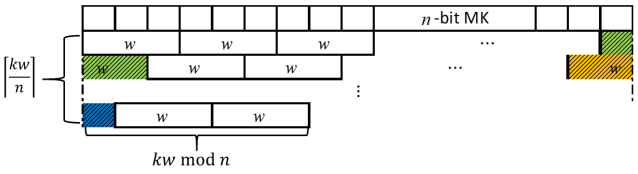

FIGURE 2. Illustration of the PKDF approach. As the random walk window, w, iterates through the MK, a bit is selected. Once this walk reaches the end of the MK, w will wrap around and continue the iterations until the desired SK length is produced [1].

the MK. The output of a single round of the PKDF is a single bit. Therefore, this process is extremely fast and can be used in a stream-like fashion to produce SKs bit-by-bit. This reduces extra overhead and allows variable-sized SKs on demand. Fig. 1 shows a general overview of this process from the PRNG output to the MK address of the SK's bit. The modulo functions allow the process to wrap around the entire length of a MK and continue the derivation. It also introduces non-linearity into the process which decreases an attacker's ability to recover internal parameters of the KDF. Fig. 2 provides a more detailed illustration of the wrapping that occurs when the end of the MK is reached during the process.

B. COMPARISON TO TRANSPORT LAYER SECURITY 1.3 HKDF

To support a deeper comparison to the existing state-of-the-art for KDF techniques, we considered comparisons with the previously mentioned HKDF, which is the key derivation function planned for implementation in Transport Layer Security (TLS) 1.3 [8]. The HKDF takes three inputs, a secret *input key material* (*IKM*), a *salt*, and an *info* string, where both the *salt* and *info* arguments are optional but can increase security and allow reuse of the *IKM*.

The HKDF is based on the HMAC defined in [9] as

$$HMAC(K, m) = H\Big((K' \oplus opad) \,||\, H((K' \oplus ipad) \,||\, m)\Big),$$

where H is a cryptographic hash function, such as Standard Hash Algorithm (SHA) variants SHA-256 [10] and SHA-3 [11] or BLAKE2 [12]; K is a secret key; and m is the message to be authenticated. $K' = K$ if the length of K <= output hash length of H, otherwise $K' = H(K)$. Both *opad* and *ipad* are constant values of 0x36 and 0x5C respectively, and $\oplus$ is the bitwise exclusive or function and $||$ denotes concatenation.

The HKDF is implemented in two phases, `Extract` and `Expand`. The `Extract` phase takes both the *IKM* and *salt* inputs and calculates an HMAC. This is to ensure that any potential weakness in the cryptographic strength of the *IKM*

TABLE 1. Summary of comparisons of PKDF and HKDF.

	PKDF	HKDF
Inputs	Master Key Window Size PRNG Seed	Input Key Material Salt Info String
Core Function	RNS-Based PRNG	Hash Function
Key Output	Bit-by-Bit	Block

is reduced. It is recommended to pass all *IKM* through the `Expand` phase regardless of strength. The result of this phase is an intermediate key value with a length equal to the hash function output (e.g., 256 bits for SHA-256). The `Extract` only requires one HMAC operation, but it may require an additional hash operation if the *salt* (HMAC key value, K) is greater than the hash digest length.

The output from the `Extract` function is then passed to the `Expand` function with the *info* string. Based on the desired length of the final derived key, this phase performs multiple rounds of the HMAC process until the concatenated outputs produce a properly sized resultant key. Therefore, unless the required key length is equal to the hash function output, extra bits will be produced that are not used (and discarded), causing an efficiency concern. For example, if the HKDF produces keys of length 42 bytes (336 bits) and uses SHA-256, then the HMAC must be calculated twice since $\left\lceil \frac{336}{256} \right\rceil = 2$. The final output from the HKDF is 512 bits long and must be truncated to the correct length.

For current information security paradigms, the HKDF is a well designed function. It offers flexibility to security with the inclusion of multiple input variables, and it is agnostic to the hash function used in the HMAC process. Therefore, its security strength is based solely on the strength of the underlying hash function. Similarly, the PKDF is built upon the security of its PRNG, but has an additional layer of protection from the KDF. However, due to the reliance on the HMAC and other design choices, the HKDF is not ideally suited for resource-limited IoT devices. Although hash functions are designed for efficiency in terms of operations, they are still relatively expensive compared to the simple design elements of the PKDF. For example, a 128-bit SK requires the HKDF to perform a minimum of 4 hash operations (plus 1 more if length of K > hash digest length). Also, the bit-by-bit key derivation of the PKDF allows greater flexibility in terms of speed and energy consumption compared to the larger block sizes of bits generated by the HMAC design and additional truncation operations to generate the correct length for shorter keys.

III. PRNG-BASED SECURITY

Based on the design of the PKDF, the main security concern should be based on the choice of the PRNG. There are many different types of PRNGs that may be implemented in devices. Linear feedback shift registers (LFSRs) are a very basic circuit that are simple, efficient, and relatively fast, which have made them popular as PRNGs, stream-ciphers, and other cryptographic primitives [13], [14], [15]. However, LFSRs are designed to be linear, deterministic, and reliant on strong seeding, creating vulnerabilities that have been compromised in past systems [16], [17]. Another popular PRNG is the Mersenne Twister [18] as it is implemented as a standard PRNG in numerous software packages. It is based on the Mersenne prime ($2^{19937} - 1$), allowing a very long period before it repeats. It is a very fast and efficient PRNG, which contributes to its popularity among numerous software distributions, such as Python and MATLAB. A recent residue number system (RNS)-based PRNG has been developed in [19] to be very efficient and allow for an exceedingly long repetition period. The design allows simple linear scaling to produce exponential increases in period length, making it an ideal candidate for resource-constrained devices. Due to in-house development of this RNS-based PRNG, it will be the main PRNG presented later in the implementations.

TABLE 2. PRNG stream output test results.

Test	15-Tap LFSR	63-Tap LFSR	Mersenne Twister	RNS-Based [19]
Frequency	100%	99.16%	99.08%	99.17%
Block Frequency	96.02%	98.81%	99.01%	99.08%
Cumulative Sums	99.79%	99.20%	99.14%	99.19%
Runs	99.86%	98.98%	98.99%	98.89%
Longest Run	94.59%	99.09%	99.13%	98.93%
DFT	83.70%	98.92%	98.97%	99.16%
Approx. Entropy	69.95%	98.79%	98.65%	98.98%
Serial	85.27%	98.92%	99.15%	99.07%

*Minimum NIST Pass Rate is 98.7% for each test.

A. NIST STATISTICAL TEST SUITE

These different PRNG stream outputs were then tested for randomness with the NIST test suite. Two different length LFSRs were evaluated to demonstrate that shorter LFSRs do not generally provide good randomness compared to longer ones. These results are shown in Table 2. As expected the 15-tap LFSR failed many of the tests (highlighted in red) indicating its lack of security as a PRNG. The Mersenne Twister also did not meet the minimum recommended pass rate for one of the tests (highlighted in yellow), but it was very close (98.65% vs. 98.7%).

In order to test the effectiveness of the PKDF, the same PRNGs were then used to produce session key streams. A random 1,024-bit MK was used to produce a stream of 80,000 128-bit SKs with each PRNG (providing over 10 million bits for statistical testing per PRNG). The generated bits were then tested with the NIST test suite for randomness. These results are provided in Table 3. The PKDF successfully masked the poor results provided from the weaker LFSR and all PRNG tests were passed.

After both of these test cases, more extensive NIST statistical tests were run to indicate other patterns and flaws in the output. All PRNGs passed except for the 15-tap LFSR. Therefore, based on these results, the PKDF provides sufficient

TABLE 3. PRNG test results with random master key.

Test	15-Tap LFSR	63-Tap LFSR	Mersenne Twister	RNS-Based [19]
Frequency	98.92%	98.90%	98.89%	98.91%
Block Frequency	98.80%	99.05%	98.97%	98.92%
Cumulative Sums	98.98%	98.87%	98.92%	98.90%
Runs	99.03%	99.05%	98.92%	99.04%
Longest Run	99.04%	98.92%	98.97%	98.94%
DFT	98.97%	98.87%	98.76%	98.84%
Approx. Entropy	98.78%	98.71%	98.92%	98.83%
Serial	98.86%	98.91%	98.93%	98.96%

*Minimum NIST Pass Rate is 98.7% for each test.

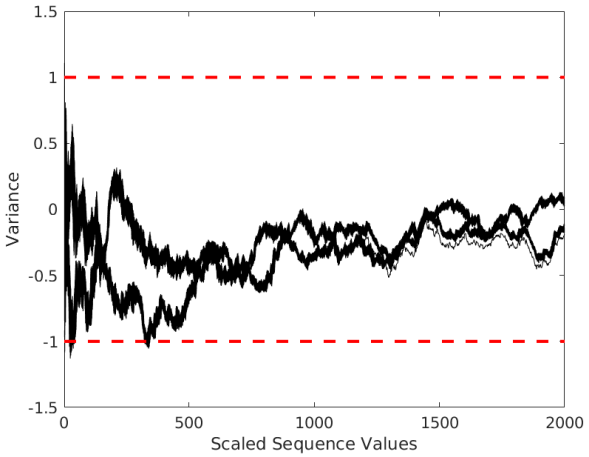

FIGURE 3. Results of LIL testing on 15-tap LFSR PKDF.

security for all the PRNGs except the 15-tap LFSR.[1] This additional layer of security in addition to the PRNG is based on the window size used and the initial starting index (both internal states to the PKDF) that must be kept secure in the case of a PRNG compromise. However, if a strong PRNG is used, such as a cryptographically secure PRNG (CSPRNG), then the exposure of the PKDF's internal state would still not allow enough information to an attacker to generate the correct derived key.

B. LAW OF THE ITERATED LOGARITHM

Although the NIST randomness test suite provides a relative level of confidence for the statistics of the output, more testing was performed based on the law of the iterated logarithm (LIL) [4] to show that the random walk variations still appear random in nature. The LIL test evaluates the fluctuations in the variance of the sequences provided from the PKDF. A sequence passes the test if it converges towards a bounds of [-1, 1] and still shows significant fluctuations. All of the PRNGs were tested with tests presented in [20]. Based on these tests, sequences pass if they are below three distance thresholds calculated by the test. As expected, the 15-tap LFSR did fall within the bounds, but it did not pass any of the thresholds because it did not have sufficient variance changes due to its short period. This failure is shown in Fig. 3,

[1]A 15-tap LFSR is a relatively weak PRNG due to its short repetition period of 32,767 bits. Therefore, in practical terms of security, this type of PRNG should never be implemented in such a design. The failure of these tests for such a weak PRNG indicates the strength of the overall PKDF.

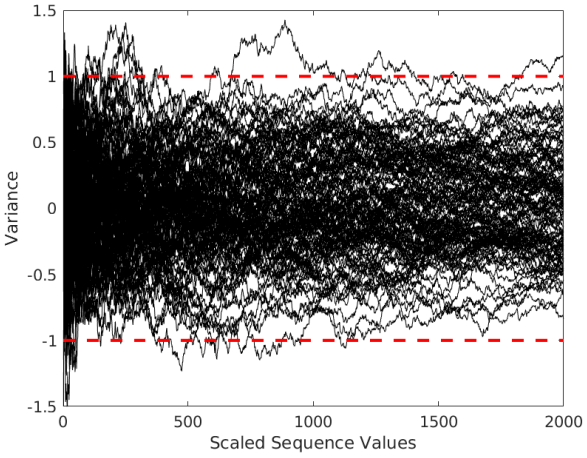

FIGURE 4. Results of LIL testing on RNS-based PKDF.

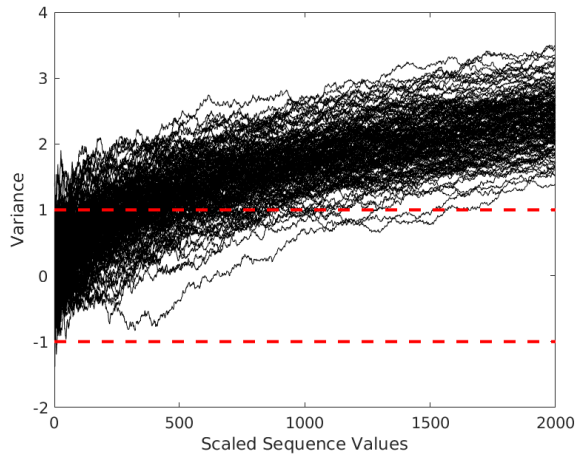

FIGURE 5. Results of LIL testing on RNS-based PKDF with unbalanced MK (513 1s, 511 0s).

and the cyclical nature of the LFSR is evident in the plot as the sequences overlap. The 63-tap LFSR passed two of the three measurement values, indicating the weakness from an LFSR. The remaining PRNGs (Mersenne Twister and RNS-based) passed all three tests without any apparent concerns. The HKDF was also tested and passed the LIL tests. Graphical results of the RNS-based PKDF are presented in Fig. 4. These results show that the PKDF has sufficient statistical variance fluctuations and does not significantly deviate from the expected bounds of $[-1, 1]$.

The LIL testing did show the PKDF is very sensitive to the count of 1s and 0s in a MK. For example, the PKDF passed when a MK was balanced in terms of equal numbers of 1s and 0s, but even with a single bit flip, the LIL test showed that the sequence variance shifted beyond the passable bounds. For example, when a MK consisting of 513 1s and 511 0s, the skew in the variance grows beyond the allowable bounds, as is displayed in Fig. 5. Fig. 6 shows the effects of different MK imbalances on the expected variance of the PKDF sequences. The top plot shows that as the imbalances grows larger, the variance is affected more. The bottom plot zooms in and shows that although the 15-tap LFSR stays

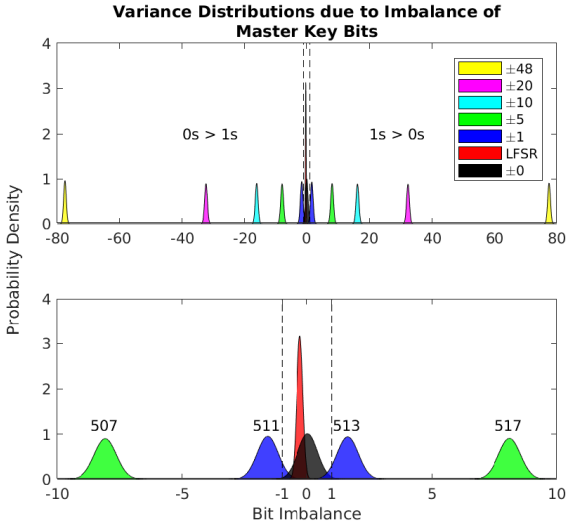

FIGURE 6. Plots of the distributions created by different imbalances in the master key's 1s and 0s. Also included is a comparison of the 15-tap LFSR that failed both the NIST and LIL testing. The top plot highlights the great effect of an unbalanced MK, and the bottom plot zooms in to show that a properly balanced MK passes the LIL tests.

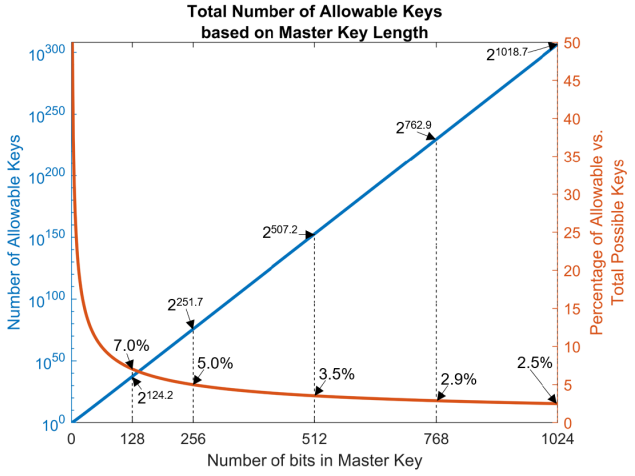

FIGURE 7. Plot of the total number of balanced keys possible for different length MKs.

within the bounds (shown with dashed lines) its variance is not sufficient due to the small period length. The PKDF is shown as the black distribution and falls perfectly inside the bounds with proper variance. Therefore, for the PKDF, a balanced MK must be used, which for a MK of a minimum length of 1024-bits reduces the number of possible MK combinations to $\binom{1024}{512}$ which is approximately 2^{1019} MKs and does not lessen the security strength significantly. The total number of balanced keys is far less than the total number of possible keys for a given length of a MK, yet the number of balanced keys still grows at an exponential rate as shown in Fig. 7.

C. JOINT AND KOLMOGOROV ENTROPIES

After testing the output of the PKDF, more analysis was needed to determine if any knowledge of bits is passed from

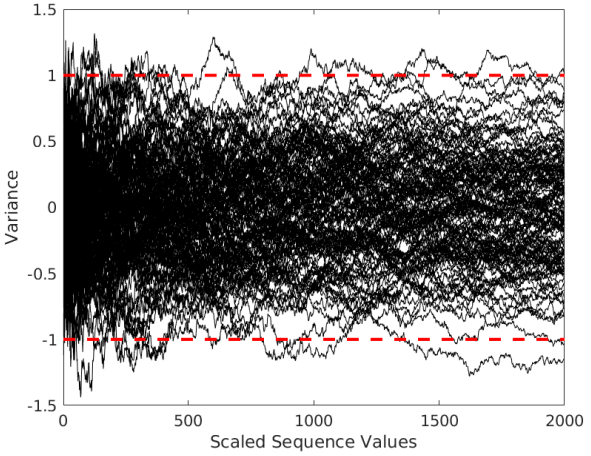

FIGURE 8. Results of LIL testing on PKDF with consideration for joint entropy.

one key to the next. This is related to the joint entropy of the two keys. Given two discrete random variables (keys) X and Y, the joint Shannon entropy is defined as:

$$H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P(x,y)\log_2[P(x,y)]$$

where x and y are values (bits) of each key and $P(x,y)$ is the joint probability of the two values [21]. Ideally, the PKDF is designed to create independent keys, therefore joint probability, $P(x,y)$ should be 0. If there is any shared information between consecutive keys, this indicates a possible vulnerability that can leak information about the key derivation process.

In order to validate the assumption that keys are independent discrete values, the XOR function was used on successive keys to create a new set of keys to be evaluated against the NIST and LIL testing from the previous sections. The new XOR keys all passed both the NIST and LIL testing indicating that there is no discernible information shared between keys from the PKDF. Fig. 8 shows the results of the LIL testing with regards to the joint entropy between keys. The sequences do not show any long term deviation of the variance that would indicate a possible deficiency in the PKDF.

Similar to joint entropy, Kolmogorov entropy indicates the relationship between session keys. For example, in the instance of a possibly cracked session key, the Kolmogorov entropy shows the impact of current key knowledge on future keys. If an attacker is able to modify and hold a single value of the PRNG, there should still be effect on the derived keys. This is shown in Fig. 9 through LIL testing, as although the number of keys are reduced, the randomness properties are not affected.

IV. SOFTWARE AND HARDWARE PERFORMANCE CHARACTERIZATION

In order to validate and characterize the performance of the PKDF, software and hardware implementations were evaluated. The software implementation was assessed on an

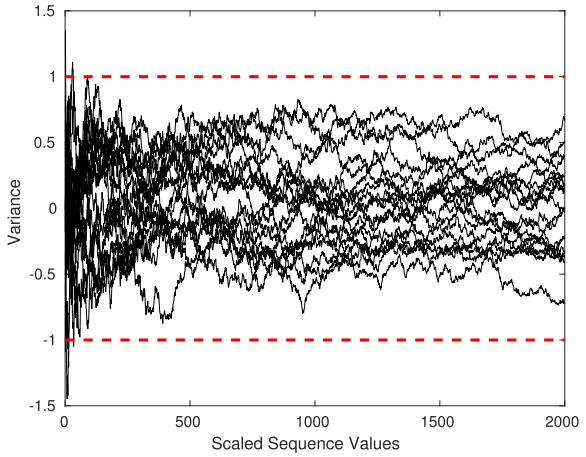

FIGURE 9. Results of LIL testing on PKDF with consideration for kolmogorov entropy.

embedded device platform to demonstrate the performance on a typical IoT-like device. It was then compared to the previously discussed HKDF design on the same platform. Next, the PKDF was designed and implemented on a field-programmable gate array (FPGA). The hardware design's goal was to determine the overall physical footprint size and achievable throughput rates.

A. SOFTWARE IMPLEMENTATION

1) MSP430

In order to accurately compare both the PKDF and HKDF performances, each one was implemented in C on a MSP430FR5994 [6]. This MSP430 device is representative of a candidate IoT edge node with limited 16-bit processing (running at 1 MHz, but maximum of 16 MHz) and memory (256 KB non-volatile, 8 KB RAM) resources. Both implementations were evaluated in total memory size, key derivation time, energy costs, and cycle counts.

The PKDF used the random walk technique with a master key of length $n = 1024$ bits and $w = 809$. A PRNG based on work in [19] was implemented as well. The memory footprint was 2,268 bytes of flash memory (ROM) and 568 bytes of RAM. Energy consumption was measured through the development kit's EnergyTrace software tool [22] by deriving 1,000 128-bit SKs from a 1024-bit MK. This derivation of keys took 52.1 seconds and 103.36 mJ of energy, resulting in an average of 52.1 ms and 103.36 μJ per key.

Next, a 128-bit SK was derived as a baseline for cycle counts. The total number of cycles required to generate a single SK was 652,693. Closer inspection of this result showed that the PRNG setup took 600,394 cycles (primarily for one-time establishment of lookup table values), while the actual derivation of the SK took only 52,211 cycles. The PRNG setup is a required cost to the PKDF, but is only called when RNS primes are changed; re-seeding initial conditions requires less than 0.6% of the total setup cost. Therefore, this setup cost can be amortized over many keys.

The HKDF was implemented using SHA-256 as the hash function due to the ease of finding optimized C libraries for embedded systems. This implementation used only a 22-byte *IKM* with no *salt* or *info* strings.[2] The baseline was performed by deriving a 256-bit SK due to the size of the SHA-256 digest. The memory footprint was 26,202 bytes of flash memory (ROM) and 182 bytes of RAM. Energy consumption was measured by deriving 1,000 256-bit SKs. The derivation of keys took 1,140 seconds and 2,108.8 mJ of energy, resulting in an average of 1.14 s and 2.11 mJ per key.

The overall key derivation required 1,139,389 cycles. Upon further breakdown, it was revealed that the hash function costs an average of 187,790 cycles per call. Based on the design of the HMAC and HKDF, the hash function is overwhelmingly the highest performance cost. Each HMAC call consists of two hash function calls. Therefore, for this baseline, three HMACs were produced, requiring a total of six hash function calls. It is also worth noting that a 256-bit block is the smallest that can be produced for this HKDF, indicating that this is the minimum number of cycles for this implementation. Moreover, due to the fact that a key is produced in multiples of the hash length, there is no amortization of hash function costs over multiple keys.

Five test cases deriving different length SKs were run to compare the performance of each KDF. The first test case derived a key of length 73 bits to highlight the scenario if a shorter (less secure) key may be needed for use in an IoT device. The second and third test case illustrate the creation of standard AES keys. The fourth test case shows the impact of the HKDF's block output design as only 8 more bits are needed for the SK, so the PKDF stays nearly constant to produce the new length, while the HKDF must produce an additional 256 bits and then truncate to achieve the new length. The final test case shows the continued scaling for the PKDF compared to the HKDF step scaling. After deriving 1,000 keys for each scenario, the average number of cycles needed to derive each key (excluding the PRNG initialization) are presented in Fig. 10. The cost of the PRNG setup is factored in to show the scalability of the PKDF highlighted in Table 4, showing the nearly constant cycles per bit of each derived key length. The slight decline also shows how the cost of the PRNG setup is amortized over many keys.

2) MSP432

After the MSP430 comparisons were completed, the disparity in the performance between the KDFs was quite large. Therefore, a second evaluation was run on a different, slightly more capable device, a MSP432P401R [7], which contains a faster 32-bit Cortex M4 processor (up to 48 MHz) and larger amounts of memory (256 KB flash and 64 KB SRAM). In order to better utilize some of these increased capabilities,

[2] Although these additional strings may be used in real applications, their use adds additional function overhead. Other scenarios were tested with different length strings for each input. As the lengths increased, the total number of cycles increased due to the hash function compressing more data. Therefore, only this scenario was chosen for best comparison performance.

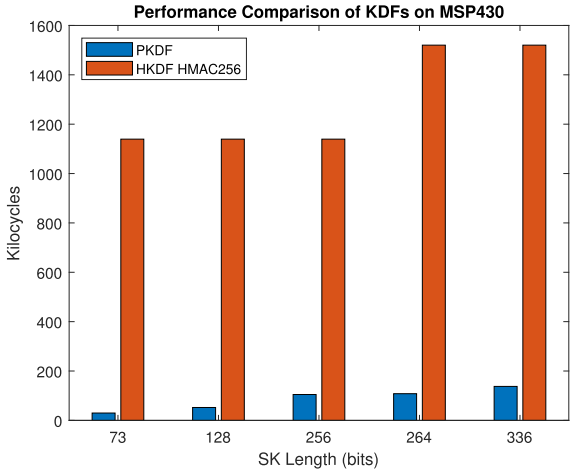

FIGURE 10. Performance comparison of KDFs as implemented on an MSP430FR5994. The SK lengths illustrate the linear scalability of the PKDF.

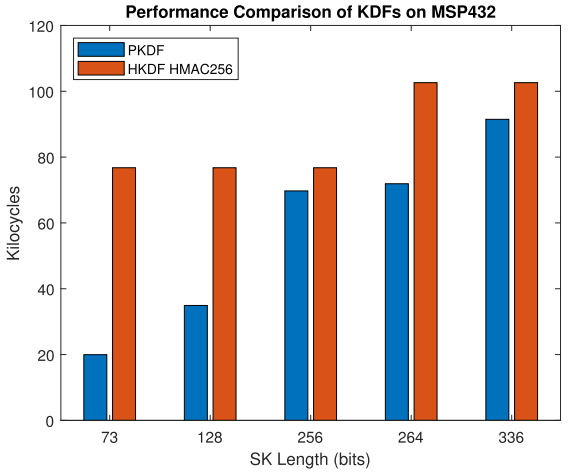

FIGURE 11. Performance comparison of KDFs as implemented on an MSP432P401R. Again, The SK lengths illustrate the linear scalability of the PKDF as was previously shown on the MSP430 comparison in Fig. 10.

TABLE 4. Scalability comparisons of KDFs on MSP430.

	Cycles/bit	
SK Length (bits)	PKDF	HKDF
73	416.0	15,608
128	412.6	8,901.5
256	411.7	4,450.7
264	411.8	5,758.4
336	411.4	4,524.4

some modifications were made to the PKDF code to allow it to perform better on the MSP432 device. However, the HKDF implementation did not require any code modifications since it was already based on optimized C libraries.

For the new PKDF implementation, the RNS-based PRNG used more residue numbers due to the increased size of available memory to store the look-up tables. This greatly increased the period[3] to approximately $2.03{\times}10^{23}$, yet the actual PRNG cycle cost greatly decreased from that used on the MSP430. Using the new values, the PRNG setup cost is 218,679 cycles, whereas it was previously approximately 600,000 for a smaller period PRNG on the MSP430. Since the PRNG is using more residue values, the total memory cost of the program grew to 16,840 bytes, yet this is still smaller compared to the HKDF implementation which was 23,024 bytes.

After the code was updated, new comparisons were run to determine the performance of both the PKDF and HKDF on the MSP432. First, similar to the MSP430, time and energy evaluations were performed on the MSP432. The PKDF was run using the same parameters ($n = 1,024$, $w = 809$) but the output key length was set to 256 bits to give a better comparison to the HKDF. For 1,000 256-bit SKs, the PKDF took 588 ms and required 17.97 mJ, resulting in an average of 588 μs and 17.97 μJ per key. The HKDF took 694 ms and required 22.08 mJ of energy in order to produce 1,000 256-bit keys, averaging 694 μs and 22.08 μJ per SK.

Next, in the same manner as the MSP430, to determine a baseline of cycle counts for key derivation, a 128-bit SK was derived using each KDF. The PKDF required 253,956 cycles in which 218,879 of those cycles was for the initialization of the PRNG. Therefore, after this sunk cost, which can be amortized over multiple keys, each 128-bit key only requires 35,077 cycles. The HKDF performance greatly increased on the more capable MSP432 reducing the total number of cycles needed to produce a 128-bit key to 76,764. Again, the hash function is the most computationally complex function of this KDF as it requires 12,213 cycles, and the minimum number of hashes for any size SK $\leq$ 256 bits is six for the HKDF.

The final comparison performed on the MSP432P401R highlights the number of cycles per bit needed to produce different key lengths. Fig. 11 shows the number of cycles needed to produce varying lengths of SKs during steady-state operation (after PRNG initialization). Although the cost of the PRNG setup can be amortized over many SKs, the cost can cause a significant increase in overall cycles. These non-amortized results are presented in Table 5. This test shows that the PKDF is nearly constant in the number of cycles per bit with a slight decrease due to the cost of the PRNG initialization being spread over more bits. The HKDF is most efficient at the 256-bit length due to the use of the HMAC256 hash function, but at smaller key length it still is less efficient compared to the PKDF.

The testing on the MSP432P401R overall showed good performance for the PKDF and substantial improvement for the HKDF compared to the evaluations on the less-capable MSP430FR5994. Much of the gains can be attributed to mature, optimized code for the architecture of the MSP432 microcontroller. Future efforts should focus on optimizing the bit-to-bit process of the PKDF to achieve increased performance. The serial nature of software limits

[3]This is a period per permutation of the RNS values. The permutation can be periodically updated to provide an entirely new sequence of outputs, decreasing the concern for overall period length of the KDF.

TABLE 5. Scalability comparisons of KDFs on MSP432.

	Cycles/bit	
SK Length (bits)	PKDF	HKDF
73	276.4	1,051.6
128	274.5	599.7
256	273.2	299.9
264	273.2	388.7
336	272.9	305.4

the ability to optimize bit-slicing techniques, yet hardware allows for better performance based on the current design.

B. HARDWARE IMPLEMENTATION

Software implementations are very flexible due to the ability to update code, but compared to hardware implementations, they are significantly slower. Hardware implementations allow the design to be highly optimized, increasing the performance gain of a function. Therefore, the PKDF was designed for implementation on a FPGA. The PKDF was initially designed in the MATLAB® Simulink® HDL Coder toolchain [23].

First, a 16-bit PRNG value, R, is passed to the function (via a register). In order to perform the modulo operation of this PRNG value, the design leveraged the use of look up tables (LUTs). The 16-bit value is then split into two 8-bit values corresponding to the upper and lower 8 bits of the PRNG value. Each 8-bit number has a 1:1 modulo value based on the modulus value, w. The LUT outputs are then summed together to return the equivalent R mod w result.

The R mod w result is then added to the previously generated value (based on random walk approach). The output of this summation is set to a size of $\lceil \log_2 n \rceil$ bits, which is equivalent to performing a mod n operation if n is a power of 2. However, if n is not a power of 2, additional memory leakage prevention logic must be incorporated to ensure the range of $[0{:}(n-1)]$ is never exceeded. After the R mod n operation, another LUT call is performed to retrieve the correct bit value of the MK. This bit value is then stored in RAM until the length of the SK is reached. An illustration of this implementation is provided in Fig. 12.

After the design was tested and correct outputs verified, the design was then ported into hardware description language (HDL) code in order to be implemented onto the FPGA. Then a core was synthesized for an Intel® Arria 10 10AS066N3F40E2SG FPGA [24]. The resulting core build required only a total of 275 adaptive logic modules (ALMs), 538 total registers, and 5 RAM blocks, resulting in approximately 0.1% overall area utilization of the Arria 10 10AS066N3F40E2SG. Based on the conversion of 1 ALM $\approx$ 2.7 logic elements (LEs) [24], a total of 743 LEs were needed in this design. As FPGA designs are not as efficient as application-specific integrated circuit (ASIC) implementations, there should only be improved performance after ASIC integrations. Therefore, this small area size is believed to make this design well suited for deployment in IoT devices.

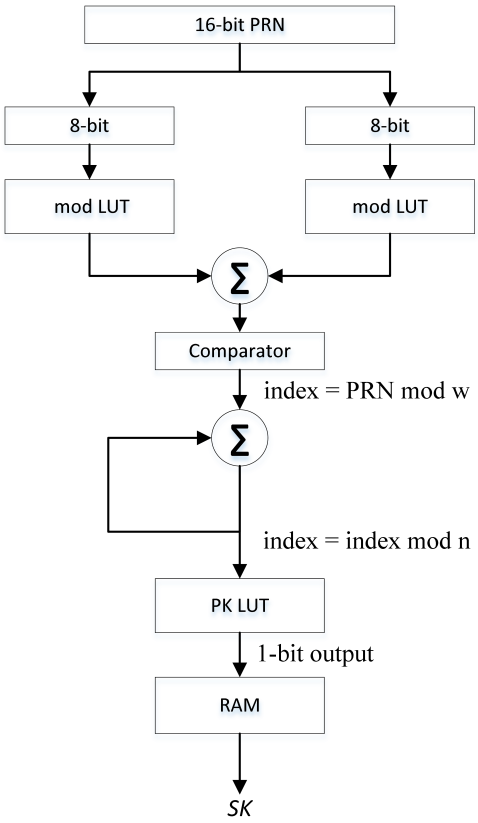

FIGURE 12. Illustration of hardware implementation of the PKDF.

The Quartus Prime build also indicated a maximum frequency, f_{max} of 469.92 MHz. Therefore, a throughput value, T, can be calculated based on the following:

$$T = \frac{k * f_{max}}{cycles}, \tag{1}$$

where k is the length of the SK and $cycles$ is the total number of cycles required to produce a SK. With $k = 128$, the total number of cycles required in this design is $k + 8 = 136$, leading to a total throughput of 442.3 Mbps per core, making multi-Gbps parallelized cores feasible. Previous work on SHA implementations led to a throughput of 1009 Mbps [25] and 1087.8 Mbps [26] per each SHA-256 hash operation. This correlates to a minimum HKDF throughput of 168.2 Mbps and 181.3 Mbps, respectively, based on the need of a minimum of six sequential hash operations for HKDF. For a low-power device implementation, this represents an approximately 2.5× multiplier of key generation throughputs.

V. CONCLUSION

This paper presented further analysis of the PKDF previously developed in [1]. This paper expanded the original idea and compared it to current technology. Comparisons to the HKDF were evaluated to show overall complexity and highlight design parameters. More complete NIST testing was completed comparing different PRNGs and the corresponding

PKDF implementation to indicate how the PKDF adds an additional layer of security on top of the PRNGs allowing the flexibility in design for resource-limited devices. Beyond the NIST testing, more mathematical testing was performed based on the law of the iterated logarithm to indicate if the PKDF process had any previously unseen weaknesses. Joint and Kolmogorov entropies were tested to show that there is no correlation between successive keys and help further quantify and validate the limited value of one cracked session key being used to break subsequent session keys. Beyond key output testing, software evaluations were performed on IoT-like devices. From this head-to-head comparison, the PKDF outperformed the HKDF in key derivation metrics such as memory footprint, cycles-per-bit, and energy consumption. On the smaller, more constrained MSP430 device, the PKDF had over an order of magnitude improvement over the HKDF in the required cycles/bit for 128-bit SK generation. Finally, a hardware design was developed and the FPGA implementation shows great promise for hardware performance.

Moving forward, more architecture and code optimization needs to be evaluated to increase the PKDF performance on more capable devices, such as the MSP432. More implementations should be designed to increase the number of platforms that can utilize the need of a fast, efficient KDF. More research needs to be done to fully implement the PKDF onto an FPGA to truly characterize the hardware performance. Following these implementations, network scalability testing should be examined with different types of devices to study the synchronization capabilities of different classes of devices. Finally, additional security options should be examined as optional features that may provide increased strength to future attacks.

REFERENCES

[1] J. M. McGinthy and A. J. Michaels, "Session key derivation for low power IoT devices," in *Proc. IEEE 4th Int. Conf. Big Data Secur. Cloud (BigDataSecurity)*, May 2018, pp. 194–203.
[2] E. Barker, *NIST Special Publication*, document 800-857, 2016.
[3] L. E. Bassham, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, N. A. Heckert, J. F. Dray, and S. Vo, "SP 800-22 Rev. 1a. A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST Publication, Gaithersburg, MD, USA, Tech. Rep. SP 800-22, 2010.
[4] Y. Wang, "The law of the iterated logarithm for random sequences," in *Proc. Comput. Complex. (Formerly Struct. Complex. Theory)*, May 1996, pp. 180–189.
[5] H. Krawczyk and P. Eronen, *HMAC-Based Extract-and-Expand Key Derivation Function (HKDF)*, document RFC 5869, Internet Engineering Task Force (IETF), 2010.
[6] *MSP430FR58xx, MSP430FR59xx, MSP430FR6xx Family User Guide*, Texas Instruments, Dallas, TX, USA, Jun. 2017.
[7] *MSP432P4xx SimpleLink TM Microcontrollers*, Texas Instruments, Dallas, TX, USA, Dec. 2017.
[8] E. Rescorla. (2015). *The Transport Layer Security (TLS) Protocol Version 1.3*. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13-26
[9] H. Krawczyk, M. Bellare, and R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, document RFC-2104, Feb. 1997.
[10] Q. H. Dang, "NIST FIPS PUB 180-4 secure hash standard," NIST Publication, Gaithersburg, MD, USA, Tech. Rep., 2015.
[11] M. J. Dworkin, "NIST FIPS PUB 202 SHA-3 standard: Permutation-based hash and extendable-output functions," NIST Publication, Gaithersburg, MD, USA, Tech. Rep., 2015.
[12] M. J. Saarinen and J.-P. Aumasson, *The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)*, document RFC 7693, Nov. 2015.
[13] K. Zeng, C.-H. Yang, D.-Y. Wei, and T. R. N. Rao, "Pseudorandom bit generators in stream-cipher cryptography," *Computer*, vol. 24, no. 2, pp. 8–17, Feb. 1991.
[14] D. V. Sarwate and M. B. Pursley, "Crosscorrelation properties of pseudorandom and related sequences," *Proc. IEEE*, vol. 68, no. 5, pp. 593–619, May 1980.
[15] S. Maitra and E. Pasalic, "Further constructions of resilient Boolean functions with very high nonlinearity," *IEEE Trans. Inf. Theory*, vol. 48, no. 7, pp. 1825–1834, Jul. 2002.
[16] D. Mukhopadhyay, S. Banerjee, D. Roychowdhury, and B. B. Bhattacharya, "CryptoScan: A secured scan chain architecture," in *Proc. 14th Asian Test Symp.*, Dec. 2005, pp. 348–353.
[17] G. Gong, S. Ronjom, T. Helleseth, and H. Hu, "Fast discrete Fourier spectra attacks on stream ciphers," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5555–5565, Aug. 2011.
[18] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, pp. 3–30, Jan. 1998.
[19] A. J. Michaels, "A maximal entropy digital chaotic circuit," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2011, pp. 717–720.
[20] Y. Wang and T. Nicol, "On statistical distance based testing of pseudo random sequences and experiments with PHP and Debian OpenSSL," *Comput. Secur.*, vol. 53, pp. 44–64, Sep. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404815000693
[21] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Hoboken, NJ, USA: Wiley, 2006.
[22] MSP. (2018). *EnergyTrace Technolog*. [Online]. Available: http://www.ti.com/tool/ENERGYTRACE
[23] MATLAB and S. Toolbox, *version 9.3 (R2017b)*. Natick, MA, USA: MathWorks Inc., 2016.
[24] *Intel Arria 10 Device Overview*, Altera, San Jose, CA, USA, Apr. 2018. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/hb/arria-10/a10_overview.pdf
[25] R. P. McEvoy, F. M. Crowe, C. C. Murphy, and W. P. Marnane, "Optimisation of the SHA-2 family of hash functions on FPGAs," in *Proc. IEEE Comput. Soc. Annu. Symp. Emerg. VLSI Technol. Archit.*, Mar. 2006, pp. 6 pp..
[26] M. Juliato and C. Gebotys, "FPGA implementation of an HMAC processor based on the SHA-2 family of hash functions," Univ. Waterloo, Waterloo, ON, Canada, Tech. Rep., 2011.

JASON M. MCGINTHY (S'18) received the B.S. degree in electrical engineering from Marquette University and the M.S. degree from the Air Force Institute of Technology. He is currently pursuing the Ph.D. degree with Virginia Polytechnic Institute and State University. His current research interest includes the energy efficient IoT security for resource constrained devices.

ALAN J. MICHAELS (SM'11) received the B.S./M.S./Ph.D. degrees in ECE, the B.S./M.S. degrees in applied mathematics, and the M.S. degree in operations research from Georgia Tech, and the MBA from Carnegie Mellon. He spent a decade at the Harris Corporation, leading research efforts in secure communications, with a particular focus in chaotic sequence spread spectrum systems. He serves as the Director for electronic systems research at the Hume Center for National Security and Technology, Virginia Polytechnic Institute and State University. He holds 41 U.S. patents.