A SPECK Based 3G Authentication Algorithm for Download onto IoT Security Modules

Keith Mayes

Information Security Group
Royal Holloway, University of London
Egham, UK
keith.mayes@rhul.ac.uk

Steve Babbage

Vodafone Group R&D
Vodafone Group Services Ltd.
Newbury, UK
steve.babbage@vodafone.com

***Abstract*—3G/4G mobile standards benefit from an authentication algorithm called MILENAGE that supports mutual authentication, protection against replay attack and generates session keys to protect confidentiality and integrity. Its usefulness attracts interest beyond the original usage, such as in securing diverse wireless and wired bearers used in Machine-to-Machine and Internet of Things (IoT) systems. The long-term reliance on one algorithm is a risk, and recently an alternative algorithm, called TUAK, was standardised as a safe-guard, should the Advanced Encryption Standard (AES) core of MILENAGE ever be found vulnerable. Previous performance evaluation of TUAK on Subscriber Identity Modules (SIM), found that it needed to be implemented in low level native code to satisfy system timing requirements; indeed this is usually the case for MILENAGE. However, deployed security modules, anticipated for the Internet of Things (IoT), generally provide access at an application layer, abstracted from the underlying hardware. Application layer implementation of TUAK was shown to be too slow to comply with standardised requirements and so an alternative faster algorithm was sought that could be downloaded and run in a compliant manner from the application level. The National Institute of Standards and Technology (NIST) has standardised a lightweight block-cipher called SPECK, and this paper describes work to create a SPECK alternative to MILENAGE and compares its performance with earlier results from TUAK.**

Keywords–3GPP; GSM; Keccak; SPECK; TUAK.

I. INTRODUCTION

This text describes follow-on work from an earlier study which evaluated the performance of the TUAK algorithm on multiple smart card platforms [1][2]. In this latest work, we sought an alternative algorithm that had credible security, yet was fast enough to be executed at a smart card application layer (rather than native code) and meet standardised performance requirements. We will start by recapping on the MILENAGE and TUAK 3G algorithms.

The Third Generation Partnership Project (3GPP) [3] has standardised an algorithm framework that permits Mobile Network Operators (MNO) to select/design their own particular cryptographic algorithms for subscriber authentication and session key generation. However, in practice, most MNOs have adopted the MILENAGE algorithm [4] that is based on AES [5]. MILENAGE is an openly evaluated and peer-reviewed example algorithm, originally designed and published by the European Telecommunications Standards Institute (ETSI) [6], Security Algorithms Group of Experts (SAGE). The security of MILENAGE is well respected, which in part accounts for its widespread use; although the use of a common approach becomes a necessity (rather than a choice) for Machine-to-machine (M2M) applications, where the particular MNO may need to change during the life of the product. Ubiquitous use of a single algorithm in equipment that may be used for many years carries a security risk, and so SAGE has standardised an alternative algorithm, called TUAK [7], which has a very different structure to MILENAGE, being built around the Keccak [8] sponge function used in the SHA3 algorithm [9].

As well as running in a system Authentication Centre (AuC), the 3G authentication algorithms must be capable of running on Subscriber Identity Modules (SIM), which are essentially smart card platforms with very restricted resources, both in terms of processor speed and available memory. It is possible to have smart cards with crypto-coprocessors, which greatly accelerate common algorithms, however these are normally not used in SIMs due to cost constraints. Therefore, SIM platforms that may offer application layer programming, typically have algorithms implemented in native code, for speed and efficiency, which are accessible via an Application Programming Interface (API). Previous work has shown that the native code approach would allow a TUAK implementation to meet the standardised performance requirements, however, application layer implementation would not. This is fine in a traditional MNO Issuer model as the native code can be defined, but it is a problem if we have to host algorithms on pre-existing general purpose SIM modules, where we only have access at the application layer.

This latter situation is increasingly likely if we now consider a future where our M2M solutions are provided in the much wider and general-purpose context of the Internet of Things (IoT). The resulting research question, is whether we can find an alternative to MILENAGE and TUAK, which meets best-practice security, yet is fast enough to be implemented at a SIM application layer and still meet the standardised performance requirements.

In Section II, an overview of MILENAGE and TUAK is provided, before introducing the SPECK algorithm in Section III. Implementation of the SPECK cipher on MULTOS is discussed in Section IV, along with some initial results. Section V describes the use of SPECK within 3G authentication, and presents the experimental results. Conclusions and suggestions for future work are offered in Section VI.

II. MILENAGE AND TUAK

In this section, we will briefly consider MILENAGE and TUAK, before suggesting how SPECK might be introduced,

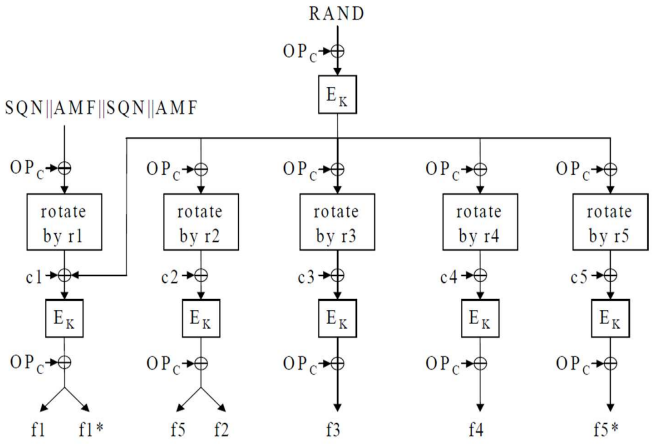

Figure 1. MILENAGE

to offer a third option for providing the mobile communication authentication and key agreement functions, specified by 3GPP for UMTS (3G) and the Long Term Evolution (LTE 4G) in [10].

A. MILENAGE

The 3G authentication solution follows a challenge-response approach similar to the earlier GSM solution, except that the challenge and responses are expanded for improved security. The structure of MILENAGE is shown in Figure 1. The challenge is 256 bits long and made up of a 128-bit random number (RAND), a 64-bit Message Authentication Code (MAC), a 48-bit sequence number (SQN) and a 16-bit management field (AMF). The MAC is recomputed on the SIM (f1 in the diagram), to ensure that the challenge was created by the genuine AuC, and the SQN is used to check that the challenge is fresh and not re-played. The result from the authentication (XRES), which is returned to the challenger, is 32-128 bits long. The algorithm generates two session keys for local storage/use, a 128-bit cipher key (CK) and a 128-bit integrity key (IK). The 48-bit anonymity key (AK) can be used during the authentication process to conceal the true value of SQN. AMF permits some home operator control of the authentication process, but is not relevant to this discussion. Within Figure 1, the repeated use of a block cipher (E_K) can be seen, which in MILENAGE is based on AES [5]. The OPc value is computed from an operator customisation value OP, however it is normal to store the pre-computed OPc within the SIM.

B. the TUAK Algorithm

The structure at the core of TUAK, as can be seen in Figure 2, is nothing like MILENAGE, as it is based on the Keccak [8] "cryptographic sponge function" [11]. This function has a good security pedigree as it is used in the SHA-3 hash function [9] and its security design properties have been investigated [12]. The authentication function is implemented by absorbing the challenge related data into the sponge, running the algorithm and then squeezing out the result and keys as shown in Figure 3. TOPc is analogous to OPc and also pre-computed and stored in the SIM; so Keccak only needs to be run twice during an authentication. TUAK uses Keccak with permutation size n = 1600, capacity c = 512 and rate r = 1088, so that only

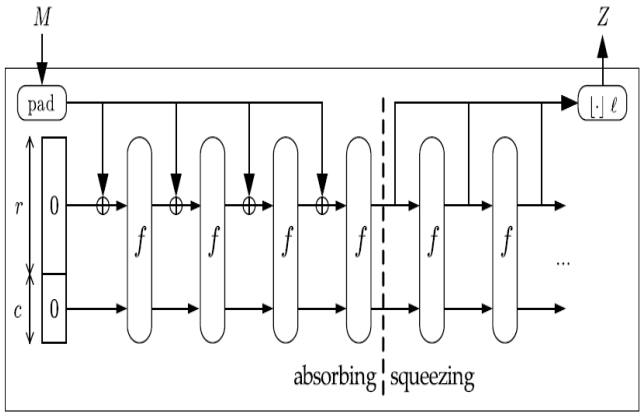

Figure 2. A Cryptographic Sponge Function

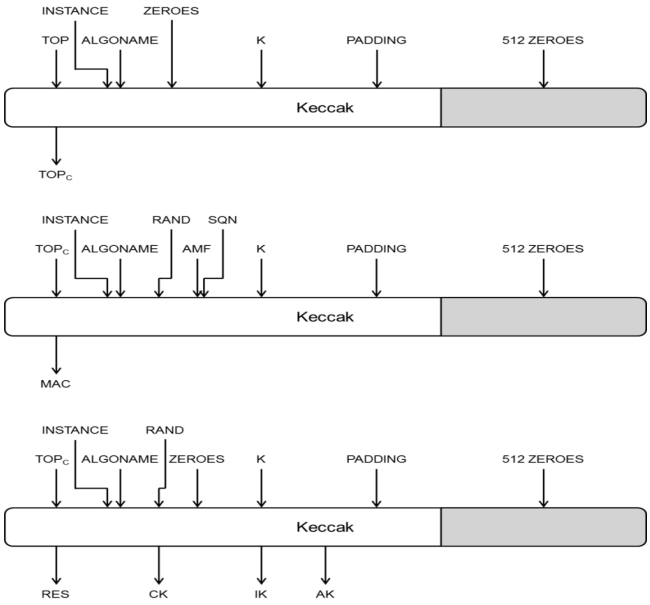

Figure 3. The TUAK Algorithm Functions

a single iteration of the permutation f is required at each stage. The feasibility of low-level native code implementation of TUAK on real secured smart card chips was first proven in [1], although this result did not extend to application level processing. To achieve the latter, requires a significantly faster core function. There are numerous candidates for such a function, however, the private research project on which this publication is based, was to specifically consider "SPECK" [13] as an alternative to the AES-based MILENAGE.

III. SPECK

SPECK is a family of block ciphers designed by a group of researchers from the NSA. SPECK, together with its sister cipher SIMON, was first published in 2013 [13], and some additional design notes were released more recently [14]. Both algorithms are designed with constrained devices in mind: SPECK is designed to have a very small footprint in software, while SIMON is designed for hardware. SPECK is a lightweight block cipher specification intended specifically for efficient implementation on resource limited chips and its performance has already been studied for use in IoT [15]. However, that performance evaluation assumed that the

TABLE I. SPECK VARIANTS

Block size 2n	Key size mn	Word size n	Key words m	Rot α	Rot β	Rounds T
32	64	16	4	7	2	22
48	72	24	3	8	3	22
	96		4			23
64	96	32	3	8	3	26
	128		4			27
96	96	48	2	8	3	28
	144		3			29
128	128	64	2	8	3	32
	192		3			33
	256		4			34

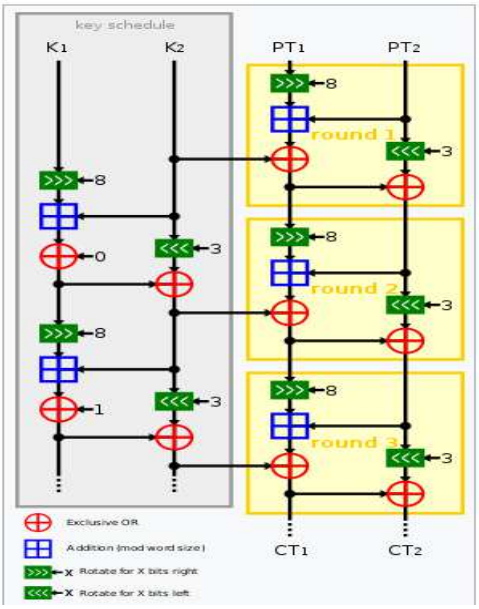

Figure 4. SPECK Overview [16]

algorithm would be native coded onto the chip, whereas here we are considering the post-issue loading of the algorithm onto a secure platform, which is abstracted from the underlying hardware and only accessible via controlled APIs.

SPECK is a family of multi-round ciphers with associated key schedules; the exact variant being defined by a number of design parameters as illustrated in Table I, which can be generally visualised in Figure 4 [16]. Note that this work found and reported an error in an earlier version of this reference diagram; which showed the K_1 XOR sequence, starting with '1' rather than '0'.

Referring back to Figure 1, the inputs and outputs to E_K are 128 bits and the minimum acceptable keysize is also 128 bits. Returning to Table I, we see that the 32 round SPECK (128/128) variant provides an ideal E_K candidate, with the 33 and 34 round being similar to the higher security modes offered in TUAK. The SPECK designers estimate that the (128/128) variant can be implemented in less than half the memory of 128-bit AES, with throughput some 70% higher. When implementing SPECK authentication, we used the same r and c parameters from the MILENAGE specification, and followed the common-practice of pre-computing OPc and storing in smart card memory

SIMON and SPECK have attracted a fair amount of attention from cryptographers, and several cryptanalysis papers have been published, attacking reduced round versions of the ciphers. Early differential cryptanalysis by Abed et al [17] and Dinur [18] performed better than simplistic exhaustive key search (albeit with a very large number of chosen plaintexts) for up to 17 rounds of SPECK; later work by Fu et al [19] extended this to 22 rounds, and then Song et al [20] to 23 rounds. Differential attacks on reduced-round versions of other members of the SPECK family have also been found by Biryukov et al [21]. Linear cryptanalysis has also been attempted, by Liu et al [22], but with less success than the differential attacks. There is thus a significant security margin between the number of rounds attacked (23) and the full number of rounds (32) in our selected variant, justifying confidence in the cipher. However, the NSA origins of SPECK and SIMON worry some cryptographers, who fear that NSA are only promoting the ciphers because they know how to break them. Notably, attempts to have SIMON and SPECK standardised by ISO have been repeatedly defeated [23]. It is not possible to determine if these suspicions have any basis in fact, so we will consider our proposal justified, based on the existing body of published security research work.

IV. Implementing the SPECK Cipher on MULTOS

Performance studies on smart cards and microcontrollers often make an assumption that an application has low-level direct access to the CPU. This is typically not the case in secure implementations, especially when the platform is intended to be strongly attack resistant. The access/interface is often via secured operating systems and virtual machines that intentionally abstract an application from the underlying hardware. MULTOS [24] cards/chips are good examples of secured platforms that work this way and results are available from the TUAK study. The SPECK MULTOS results will provide an interesting comparison to the native coded implementation in the later stage of the study. It is also worth noting that MULTOS has been proposed for use in IoT due to its capability for secure application loading/management (PKI based) in the field. The implementation approach for MULTOS is to initially develop the SPECK code on a simulator and then load onto a real card for performance measurement. We will start with a naive implementation based on published pseudo code to first check compliance with the associated test vectors. We will then refine/optimise the implementation as we progress through the study. We know from previous work that MULTOS performance struggles when a variable parameter is used to control the extent of block shifts/rotates, and so it is suspected that the naive implementation of SPECK will be slow, as it follows the pseudo code and specifies the shift amounts as variables. Once we have confidence in the core SPECK implementation we will fit it within the 3G authentication framework for the comparative (with TUAK) performance tests.

A. Initial Results/Observations

Considering the simplicity of the SPECK cipher there were some unexpected difficulties in creating a version which produced the correct test vector result. Firstly, the 'C' code examples for the 128/128 mode use 64-bit unsigned integers (uint64_t) that are not universally supported by 'C' compilers, especially for legacy and specialist secured microcontrollers. This meant that the pseudo code could not be used unmodified as the starting point. Of course as the target CPU is 16-bit, the

TABLE II. INITIAL MULTOS FUNCTIONAL TEST RESULTS

Test Type	Execution Time (ms)	Comment
Null	43	Command handling and 16 bytes of data in message response, with no actual function performed. This has already been subtracted from other results
SPECK	253	Full cipher 128/128 functionally correct and producing correct test vector result [No MULTOS primitives]
SPECK	157	Uses pre-computed key-schedule (256 bytes) [No MULTOS primitives]
Rotate Right 8 places	1.01	[No MULTOS primitives]
Rotate Left 3 places	1.84	[No MULTOS primitives]
Add 64 bit results	1.25	[No MULTOS primitives]

compiler would just have to mimic the manually coded creation of the 64-bit types. For convenience and efficiency, a union type was created so that the 64-bit storage could be accessed as constituent 32, 16 and 8 bit unsigned integers. In order to obtain intermediate round and key-schedule reference results it was decided to first implement the example code using the lcc compiler on a Windows PC and dump intermediate results to a log file, for later use in verifying the MULTOS implementation. Once the reference test version was working and the test results produced, the equivalent functionality was coded for the MULTOS platform. The initial style was reasonably efficient, avoiding unnecessary loops, function calls and stack-based parameter passing; and speed critical variables were created in reserved session RAM. The latter is important as non-volatile storage on the target microcontrollers is significantly slower than RAM. However, at this stage we did not use any MULTOS primitives so all the functionality was coded at the application layer. We did however provide an option to use a pre-computed key schedule as discussed below.

1) Key-schedule Pre-computation: In smart card devices it is quite normal to store a few kilobytes of sensitive account specific data (e.g., certificates, keys, photos, account details, IDs, PINs etc.) during the personalisation process. This data is stored in the non-volatile memory (characterised by fast reads and slower writes) that is shared with the code. For the SPECK cipher mode under consideration (128/128), we must at least store a 128bit (16 byte) long-term secret key 'K' . The key-schedule requires a 64 bit (8 byte) key for each round that is generated from 'K'. For a non-optimised solution the round function for round key generation is the same as for the data processing, so no significant extra code space would be required. An optimised version (avoiding parameter passing) would likely need some additional code space. Pre-computation of the key schedule would require 32x8 = 256 bytes of non-volatile memory which is by no means prohibitive, even in old smart cards. As Round is called almost as many times for round key generation as for the data processing, there is potential for significant speed improvement by using pre-computation.

Once the MULTOS code was functional it was loaded onto an ML3-80K-R1 MULTOS test card, which is based on the same Infineon SLE78 chip [25] that we used in the TUAK performance study. Commands were then sent to the card using the MULTOS scripting utility (MUTIL) and response times recorded. Table II, gives a summary of the first functional test results.

Often encoding an algorithm at the MULTOS application layer results in an implementation that is too slow to be practical; as was the case with TUAK. Although 253ms is not fast for running a block cipher, it is not necessarily unusable and there is still the potential for speed-up by exploiting suitable MULTOS primitives. Furthermore, pre-computation and personalisation of long-term round keys is quite practical and so the 157ms execution time is considered the benchmark performance at this stage. We will explore the use of primitives in the next stage of the study when we will see if SPECK could satisfy the SAGE/GSMA performance requirements on a MULTOS platform.

TABLE III. RESULTS WITH AND WITHOUT PRIMITIVES

Test Type	No Primitives Execution Time (ms)	Primitives Execution Time (ms)
Null	43	43
SPECK (precomputed key-schedule)	154.57	84.34
Rotate 64 bits Right by 8 places	1.01	0.63
Rotate 64 bits Left by 3 places	1.84	0.83
Add 64 bit inputs	1.25	0.54
XOR 64 bit inputs	0.56	0.53

V. IMPLEMENTING 3G SPECK AUTHENTICATION ON MULTOS

The results so far have shown better performance than had been anticipated on the MULTOS platform, however they did not make use of standard primitives offered by the platform. Use of such primitives is advisable for performance, but also for security reasons as their implementation is likely to have been defensively coded and evaluated against attack

A. 4.1 Conversion to MULTOS Primitives

The SLE78 is an innovative security controller intended for high security applications. Instead of relying mainly on shields and sensors it uses "Integrity Guard", which exploits dual CPUs working in tandem. The supported primitives of main interest included:

- multosBlockShiftRight()
- multosBlockShiftLeft()
- multosBlockAdd()
- multosBlockXor()

Note that MULTOS does not offer block rotates, however these are simple to implement using the shift primitives. Prior to converting the initial code to make use of primitives and adding support for 3G authentication processing, some test commands were created to practically determine performance impact. The set of results are presented in Table III, for the case of a pre-computed key-schedule. Note that all the primitive test utilities also made use of multosBlockCopy().

Considering Table III, with the exception of XOR, all the basic functions roughly doubled in speed when using platform primitives; notably SPECK execution time reduced from 154.57ms to 84.34ms. XOR is a very simple function at the application layer and so little improvement (0.56ms to 0.53ms) was anticipated. The notable effect is that using the available primitives nearly doubles the overall speed of the SPECK block cipher. This is an interesting result for the 3G authentication comparison, as following the 3G MILENAGE structure, requires the block cipher to be called multiple times.

TABLE IV. MILENAGE FUNCTION MAPPING

Function	Mapping
f1	MAC-A 64 bits
f1*	MAC-S (alternative used in re-synch)
f2	RES 64 bits
f3	CK 128 bits (ciphering key)
f4	IK 128 bits (integrity key)
f5	AK 48 bits (anonymity key)
f5*	Alternative AK used in re-synch

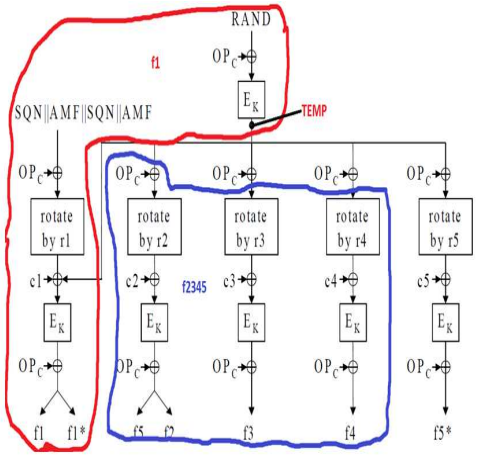

Figure 5. Implementation split of f1()and f2345()

B. Implementation of 3G Authentication

Referring back to Figure 3, we will adopt the same implementation approach as used for the earlier TUAK work so the results can be as closely comparable as possible. This requires development of two functions making use of the SPECK block cipher within the MILENAGE structure. The functions are separated in this way as they have different input values.

- SPECK_f1() computes f1 (and f1*)
- SPECK_f2345 computes f2, f3, f4, f5

The output of f5* is not part of a regular authentication vector (it is only used when resynchronisation of the sequence number SQN is needed), and so will not be included in our measurement. For clarity we present the mapping of these functions to the protocol, within Table IV.

We have reproduced the MILENAGE structure in Figure 5; showing the practical split of the f1 and f2345 functions.

When testing, it is assumed the f1() is run first so that the value TEMP can be re-used in f2345(), avoiding an extra encryption. Within f2345(), TEMP only needs to be XORed once with the (pre-computed) OPc. Considering the operations in addition to ciphering, we can see rotates and XORs using standardised values. Fortunately the rotates are easy when using the standard r values; r1 = 64, r2 = 0, r3 = 32, r4 = 64 (and r5 = 96). The integers are all multiples of 8 bits and so rotates can be performed simply with byte copies. The XORs with the c constants are also very simple as the values only affect the least significant byte, so byte, rather than block XORs are needed. The functions were coded using MULTOS primitives and the initial results are presented in Table V.

It can be seen from Table V, that the function times are dominated by the SPECK block cipher execution times. The total time (433.8ms) is very interesting as it fits within the

TABLE V. AUTHENTICATION FUNCTION EXECUTION TIMES (ms)

Function	Execution Time (ms)	Block Encryptions
f1()	173.16	2
f2345()	260.64	3
Total	433.80	5

TABLE VI. MULTOS TUAK AND SPECK AUTHENTICATION COMPARISON

Function	TUAK Authentication (ms)	SPECK Authentication (ms)
f1()	1529	173.16
f2345()	1575	260.64
Total	3104	433.80

revised performance specification proposed to SAGE during the TUAK study [26].

..."The functions f1-f5 and f1 shall be designed so that they can be implemented on a mid-range microprocessor IC card (typically 16-bit CPU), occupying no more than 8 kbytes non-volatile-memory (NVM), reserving no more than 300 bytes of RAM and producing AK, XMAC-A, RES, CK and IK in less than 500 ms total execution time."...*

Although the test implementation has no added high-level measures for defensive coding, the application is running on a MULTOS platform, which is marketed for its evaluated high-security capabilities. It would be expected that the platform's chip and native code would have defensive measures to counter fault and side-channel attacks. The fact that a platform level implementation can satisfy performance constraints suggests that the functionality could be added to stock or issued devices. This is a major operational advantage that was not available for the TUAK implementation. Although Java Card [27] is out of scope for this study, it is not unreasonable to consider that performance might also be adequate on that platform type; although it is possible that added defensive measures might be needed at the application layer.

C. MULTOS Comparison of SPECK with TUAK Authentication

A main purpose of this study was to see whether TUAK could be used as an alternative to SPECK in Internet of Things (IoT) type applications, where the processors have limitations similar to those of traditional smart cards. To measure this, the SPECK authentication functionality has been implemented in the same manner and on the same platforms as in the earlier TUAK study, so results are directly comparable. It may be recalled that TUAK did not suit MULTOS application implementation using existing primitives, as the comparison in Table VI shows. In fact, the TUAK authentication takes roughly 7x as long as SPECK.

MULTOS chips are being proposed for use in IoT applications and the results suggest that they could support SPECK on existing/standard platforms, whereas a custom primitive is essential for TUAK.

D. Native mode Comparison of SPECK with TUAK Authentication

Although this study was primarily focussed on downloading and running an authentication algorithm on a platform application layer, a native mode implementation was also created for the Samsung S3CCE9E4/8 chip (more detail in

TABLE VII. NATIVE MODE TUAK AND SPECK AUTHENTICATION COMPARISON

Function	TUAK Authentication (ms)	SPECK Authentication (ms)
f1()	77.88	3.35
f2345()	78.18	5.13
Total	156.06	8.48

[2]), which is a 16-bit secured smart card chip also used in the earlier TUAK performance study. For interest, the comparative results are presented in Table VII.

Clearly the native SPECK implementation is very fast (8.48ms) even on an old chip, but the result should be viewed with caution as there are no defensive coding measures in place, which could easily add an order of magnitude to the execution time.

VI. Conclusions and Future Work

The experiments conducted during this project have shown that the SPECK cipher is significantly faster than Keccak (TUAK core) when implemented on the MULTOS platform (or indeed in native mode). The results from the MILENAGE style authentication using SPECK, were significant as they showed that a MULTOS application layer implementation could satisfy the 3GPP timing constraints; something that was not possible with TUAK. There would even be headroom to add a few extra rounds to the SPECK function to increase its security margin, if preferred. This means that the algorithm could be loaded and configured post-issue, which has relevance for M2M and general IoT devices. In fact, MULTOS has begun to position itself in IoT because apart from its secure design, it also has an application loading mechanism based on PKI. This means devices can be configured offline, outside of secure environments, and without the need to distribute shared secret keys. 3GPP does not specify a standard algorithm, but rather a framework with MILENAGE and TUAK presented as example implementations; and so the SPECK version could be used within the standard. For the future, it would be interesting to implement the SPECK-based 3G authentication on a Java Card and compare the results with the MULTOS platform results.

Acknowledgment

The authors would like to thank the UK National Cyber Security Centre (NCSC) for its support of this work.

References

[1] K. Mayes, S. Babbage, and A. Maximov, “Performance Evaluation of the new TUAK Mobile Authentication Algorithm”, in Proc. ICONS/EMBEDDED, pp. 38-44, 2016.

[2] K. Mayes and S, Babbage, “A Multi-Platform Performance Evaluation of the TUAK Mobile Authentication Algorithm”, International Journal on Advances in Security, vol 9, no. 3&4, pp. 158-168, 2016.

[3] (2017, Oct.) The Third Generation Partnership Project website, [Online]. Available: http://www.3gpp.org/, [retrieved: March, 2018].

[4] 3GPP TS 35.206: 3G Security; Specification of the MILENAGE algorithm set: An example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 2: Algorithm specification (2014).

[5] Federal Information processing Standards, Advanced Encryption Standard (AES), FIPS publication 197 (2001).

[6] (2017, Oct.) The European Telecommunications Standards Institute website, [Online]. Available: http://www.etsi.org/, [retrieved: March, 2018].

[7] 3GPP, TS 35.231: 3G Security; Specification of the TUAK algorithm set: A second example algorithm set for the 3GPP authentication and key generation functions f1, f1*, f2, f3, f4, f5 and f5*; Document 1: Algorithm specification (2014).

[8] G. Bertoni, J. Daemen, M. Peeters, and G. van Aasche, “The keccak Reference”, version 3.0, 14 (2011).

[9] NIST, Announcing Draft Federal Information Processing Standard (FIPS) 202, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, and Draft Revision of the Applicability Clause of FIPS 180-4, Secure Hash Standard, and Request for Comments, (2004).

[10] 3GPP, TS 33.102: UMTS 3G Security; Security Architecture, V11.5.1 (2013).

[11] G. Bertoni, J. Daemen, M. Peeters, and G. van Aasche, “Cryptographic Sponge Functions”, version 0.1, (2011).

[12] G. Gong, K. Mandal, Y. Tan, and T.Wu, “Security Assessment of TUAK Algorithm Set”, [Online]. Available: http://www.3gpp.org/ftp/Specs/archive/35_series/35.935/SAGE_report/Secassesment.zip (2014), [retrieved: March, 2018].

[13] R. Beaulieu, et al, The SIMON and SPECK Families of Lightweight Block Ciphers, Cryptology ePrint Archive, Report 2013/404, 2013, [Online]. Available: http://eprint.iacr.org/2013/404, [retrieved: March, 2018].

[14] R. Beaulieu, et al, Notes on the design and analysis of SIMON and SPECK, Cryptology ePrint Archive: Report 2017/560, 2017, [Online]. Available: http://eprint.iacr.org/2017/560, [retrieved: March, 2018].

[15] R. Beaulieu, et al, Simon and Speck: Block Ciphers for the Internet of Things, National Security Agency, (2015).

[16] (2017 Oct.) Wikipedia, Speck (cipher), [Online]. Available: https://en.wikipedia.org/wiki/Speck_(cipher), [retrieved: March, 2018].

[17] F. Abed, E. List, S. Lucks, and J. Wenzel, Cryptanalysis of the speck family of block ciphers, Cryptology ePrint Archive, Report 2013/568, 2013, [Online]. Available: http://eprint.iacr.org/2013/568, [retrieved: March, 2018].

[18] I.Dinur, Improved Differential Cryptanalysis of Round-Reduced Speck, Cryptology ePrint Archive: Report 2014/320, [Online]. Available: http://eprint.iacr.org/2014/320, [retrieved: March, 2018].

[19] K.Fu; et al, MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck, Cryptology ePrint Archive: Report 2016/407, [Online]. Available: http://eprint.iacr.org/2016/407, [retrieved: March, 2018].

[20] L.Song, Z.Huang, and Q.Yang, Automatic Differential Analysis of ARX Block Ciphers with Application to SPECK and LEA, Cryptology ePrint Archive: Report 2016/209, [Online]. Available: http://eprint.iacr.org/2016/209, [retrieved: March, 2018].

[21] A.Biryukov, A.Roy, and V.Velichkov, Differential Analysis of Block Ciphers SIMON and SPECK, Cryptology ePrint Archive: Report 2014/922, [Online]. Available: http://eprint.iacr.org/2014/922, [retrieved: March, 2018].

[22] Y.Liu, et al, Linear cryptanalysis of reducedround SPECK, Information Processing Letters Volume 116, Issue 3, March 2016.

[23] CNBC News, Distrustful US allies force spy agency to back down in encryption row, September 2017, [Online]. Available: https://www.cnbc.com/2017/09/21/distrustfulusalliesforcensatoback-downinencryptionrow.html, [retrieved: March, 2018].

[24] (2017, Oct.) MULTOS website, [Online]. Available: http://www.multos.com/, [retrieved: March, 2018].

[25] (2017, Oct.) Infineon, SLE 78 family of 16-bit security controllers, [Online]. Available: https://www.infineon.com/cms/en/product/security-and-smart-card-solutions/security-controllers/sle78/channel.html?channel=5546d462503812bb015066c2d8e91745, [retrieved: March, 2018].

[26] (2017, Oct.) K. Mayes, ”Performance Evaluation of the TUAK algorithm in support of the ETSI Sage standardisation group”, 3GPP, 2014, [Online]. Available: at http://www.3gpp.org/ftp/Specs/archive/35_series/35.936/SAGE_report/Perfevaluation.zip, [retrieved: March, 2018].

[27] Oracle, Java Card Platform Specifications V3.04, (2011).