Performance Evaluation of the new TUAK Mobile Authentication Algorithm

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

Alexander Maximov
Ericsson Research
Ericsson
Lund, SE
alexander.maximov@ericsson.com

***Abstract*—TUAK is a new mutual authentication and key generation algorithm proposed by the Security Algorithm Group of Experts (SAGE) of the European Telecommunications Standards Institute (ETSI) and published by the Third Generation Partnership Project (3GPP). TUAK is based on the Keccak sponge function which has very different design principles to the pre-existing 3G MILENAGE algorithm and so promises a back-up/alternative in case algorithm vulnerabilities are discovered during long-term Machine-to-Machine (M2M) deployments. However, the practicality of implementing TUAK on currently deployed and/or future Subscriber Identity Module (SIM) cards is not well known. This paper describes the initial work and findings of a study in support of SAGE and GSMA to consider such implementation aspects.**

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

I. INTRODUCTION

The European Telecommunications Standards Institute (ETSI) [1] and later the Third Generation Partnership Project (3GPP) [2] standardised mobile networks so that Mobile Network Operators (MNO) were able to choose/design their own cryptographic algorithms for subscriber authentication and session key generation. In GSM, [3] there is a proliferation of algorithms, however for 3G most MNOs use the well-studied and openly published MILENAGE algorithm [4]. MILENAGE (AES [5] based) was designed and published by the ETSI Security Algorithms Group of Experts (SAGE), and more recently SAGE designed a second algorithm, called TUAK [6] based on the Keccak [7] sponge function. This was done for two main reasons. Firstly, although MILENAGE is currently considered strong, industry should have a proven alternative in case an advance in cryptanalysis exposes vulnerability. Secondly, machine-to-machine (M2M) devices will use "embedded SIMs", whereby a Subscriber identity Module (SIM) chip is fitted into a device, and the assignment (or re-assignment) to a MNO and the provisioning of security credentials is done later, over the air. Some devices may be deployed for at least twenty years, which is a considerable time in the life of a technical security solution. Having two strong algorithms (MILENAGE and TUAK) built into the hardware, and available for selection, should give good assurance that effective security can be maintained throughout the SIM lifetime.

TUAK inherits most of its security characteristics from Keccak, which is the winning SHA-3 design and has of course been extensively studied. See [8][9] for a closer analysis of TUAKs security. TUAK is fundamentally different from MILENAGE in its design, so that an advance in cryptanalysis affecting one algorithm is unlikely to affect the other. There are very few academic publications around TUAK as the standards are quite new, although a comprehensive security assessment [10] of the TUAK Algorithm Set was carried out by the University of Waterloo, Canada. It considered a wide range of cryptanalysis techniques, and finally concluded that TUAK can be used with confidence as message authentication functions and key derivation functions. However, industry acceptance and adoption of TUAK requires not just a secure design, but also confidence that it can be implemented on limited resource SIMs with sufficient performance.

- Is it possible to load the algorithm onto an existing deployed or stocked smart card platform?
- If so, will the algorithm run with acceptable performance?
- Will a new SIM require a crypto-coprocessor for adequate performance?
- Will a new SIM need to have a high performance processor (e.g., 32-bit type)?
- Will a new SIM require specialist low-level software for the algorithm?
- Will the algorithm benefit from hardware security protection?

There have been previous performance evaluation and comparisons [7][11][12], around the Keccak core for the SHA-3 competition [13], however these were aimed primarily at specialist hardware, or far more powerful and less memory limited processors than are typically found in SIMs. Therefore, at the request of SAGE, the evaluation described in this paper was undertaken, in which the entire TUAK algorithm performance was determined by experiment with the SAGE specified settings for Keccak, using their published source code as a starting point. The latter is important, as SIM vendors tend to base their implementations on the published security standards examples. In addressing the performance questions it was necessary to define a method of experimentation that would give relevant results yet would not be tied to a particular processor, platform or optimised for particular chip features. The work began with the PC example implementations, before forking to a parallel development suited for smart card evaluation. For the latter, simulation was originally considered, however it is difficult to map results to real card performance. The use of a multi-application card platform was included as a positive means of abstraction from any particular chip, and

 ISBN: 978-1-61208-451-0

could be representative of loading the algorithm onto existing/stock SIMs. However, the performance of such platforms (e.g., MULTOS [14]/Java Card [15]) is usually inferior to a native card implementation and so native mode was included as the principal benchmark.

In Section II an overview of TUAK is provided before describing the experimental setup and software development in Section III and Section IV. Results are presented in Section V and analysed in Section VI. Some comments on security defences and performance are discussed in Section VII and finally, conclusions and future work are presented in Section VIII.

II. TUAK Overview

In each of GSM/GPRS (2G), UMTS [16] (3G) and the Long Term Evolution (LTE 4G), a fundamental part of the security architecture is a set of authentication and key agreement functions [17][18]. The set of functions varies between generations, with 3G providing more security than 2G, and 4G adding some further refinements. These functions exist in the subscriber's SIM card (which is provided by their MNO), and in a network node called the Authentication Centre (AuC) that is run by the MNO. The 3G authentication and key agreement architecture requires seven cryptographic functions. MILENAGE [4] is a complete set of algorithms to fulfil these functions, built from a common cryptographic core (the AES block cipher) using a consistent construction. TUAK [6] is also a complete set of cryptographic functions for 3G authentication and key agreement. LTE security reuses the same set of functions, so both MILENAGE and TUAK can also be used for LTE. There is also a standardised method for using the 3G authentication and key agreement functions in GSM/GPRS.

A. Algorithm Inputs and Outputs

Whereas MILENAGE was designed only with 3G in mind, TUAK was also designed for LTE and so supports a 256 bit subscriber-unique secret key as well as the 128 bit key size used in 3G. Moreover, TUAK also allows for the possibility that certain other input or output parameters might increase in length in the future. The input and outputs of TUAK's seven cryptographic functions *f1, f1*, f2, f3, f4, f5 and f5** are defined in [6] and like MILENAGE, the TUAK algorithm-set expects one additional input parameter, an "Operator Variant Algorithm Configuration Field". In the case of TUAK, this field is called TOP and is 256 bits long; each mobile operator is expected to choose its own value for this, typically the same value for many SIMs. The 3GPP security architecture did not require this extra parameter, but it was included for two main purposes:

- SIMs for different MNOs are not interchangeable, either through trivial modification of inputs and outputs or by reprogramming of a blank SIM.
- By keeping some algorithm details secret, some attacks (such as side channel attacks like power analysis) become a *little* harder to carry out.

TUAK includes an algorithm to derive value TOP_c from TOP and the secret key K, and it is sufficient for the SIM card to be programmed with TOP_c rather than with TOP itself. This means that an attacker who is able to extract TOP_c from one card does not learn TOP or TOP_c for other cards.

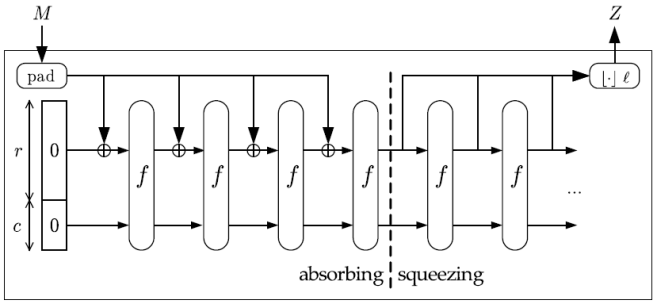

Figure 1. A Cryptographic Sponge Function

B. Algorithm Building Blocks

The main building block from which all of the TUAK algorithms are constructed is Keccak [7], the "cryptographic sponge function" which was selected by NIST as the winner of the SHA-3 hash function competition [13]. Sponge functions work by repeated application of a fixed length transformation or permutation *f*, as shown in Figure 1, which is copied from [19]. First the input bits are "absorbed", and then the output bits are "squeezed out".

TUAK uses the Keccak algorithm with permutation size $n = 1600$, capacity $c = 512$ and rate $r = 1088$. This rate value is big enough that each of the algorithms in the TUAK set needs only a single instance of the permutation f - repeated iteration of the permutation is not necessary.

Details of the TUAK algorithm can be found in [6], with test data in [20][21]. The TUAK algorithm functions are illustrated in Figure 2. In this diagram:

- The top picture shows how TOP_c is derived from TOP.
- The middle picture shows how MAC-A or MAC-S is computed (*f1* and *f1**)
- The bottom picture shows how RES, CK, IK and AK are computed (functions *f2, f3, f4, f5* and *f5**) - note that these functions all take exactly the same set of input parameters, so can be computed together
- INSTANCE is an 8-bit value that takes different values for different functions, for different input and output parameter sizes, and to distinguish between *f1* and *f1** and between *f5* and *f5**, providing cryptographic separation
- ALGONAME is a 56-bit ASCII representation of the string "TUAK1.0"
- The block labelled "Keccak" is the 1600-bit permutation, with the shaded part corresponding to the 512-bit "capacity" input; see Figure 2.

III. The Experimental Setup

Based on the arguments presented in the introduction, the goal was to use a combination of PC software, native smart card chip implementations and a secure platform for development and comparative testing. For the native implementations, we required two chips of comparable CPU power, yet different security protection to determine if the inherent protective measures impacted performance. Furthermore, to

 ISBN: 978-1-61208-451-0

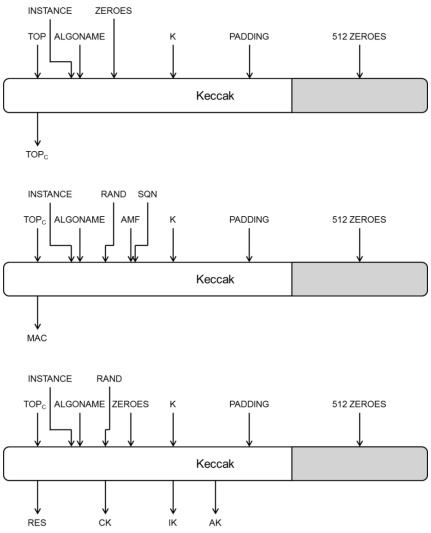

Figure 2. The TUAK Algorithm Functions

make useful comparisons with the secure platform implementations, we needed platforms based on similar chips. A solution presented itself based around native implementations on the Infineon SLE77 [22] and SLE78 [23]. The MULTOS platform was selected as the secure platform primarily because test cards (types M3 and M4) were available based on the same Infineon chips. The smart card experiments were preceded by measurements on a PC platform that used similar example C code. The code could in future also be ported to Java Card platforms, although the Java coding language would make comparisons less clear.

A. The PC Test Platform

The initial tests used an Intel Core i5-2540M CPU @ 2.60GHz, max turbo frequency 3.30GHz, 2 cores, 4 threads, Intel Smart Cache 3Mb, Instruction set 64-bit + AVX, 4Gb RAM, with a Windows 7 32-bit OS. The Keccak example implementations were written in C and compiled to optimise speed. Although the processor and the compiler supported 64-bit integers, the resulting assembly code was limited by the OS to 32-bit. Execution time was measured in CPU clock cycles, although multiple runs were necessary due to the multi-tasking OS interrupting execution. Various versions of the example code became available during development as shown in Table I. The smart card source code was originally modelled on version 1 and then developed in parallel.

In Keccak, f is a permutation. Keccak is a family of algorithms, from which a particular algorithm is selected by setting three security parameters:

- The permutation size n, which can be 25, 50, 100, 200, 400, 800 or 1600 bits.
- The "capacity" c, which is a security parameter (essentially, for a given capacity c; Keccak is claimed to stand any attack up to complexity $2^{c/2}$).
- The "rate" $r = n - c$, which determines how many input and output bits can be handled by each iteration of the permutation.

TABLE I. PC EXAMPLE CODE IMPLEMENTATION VERSIONS

Version	SupportedBits	ShortDescription
0	8/16/32/64	Size optimized, generic, use of % and more tables
1	8/16/32/64	Speed optimized, generic
2	64	Use of CPU 64-bit rotate instruction
3	8/32/64	Original from the specification
4	64	Similar to v2 but trying to combine more operations
5	32	Totally unrolled version, only C code
6	8/16/32	With bit-interleaving, generic, not optimized
7	32	Optimized bit-interleaving, part unrolled, 32-bit

B. The Smart Card Chips

The chips for experimentation both had 16-bit CPUs, which is a size representative of the majority of deployed SIMs (although there are still 8-bit CPUs around, as well as newer 32-bit CPUs). Whilst they are of similar family, horsepower and vintage they are quite different in security aspects.

1) SLE77: The SLE77 is a traditional style security controller intended for mid-range payment applications, and evaluated to Common Criteria [24] EAL5+. Its crypto-coprocessor does not support TUAK/Keccak so was not used in our tests. Details of the chip protection measures against physical, side-channel leakage and faults are not publicised, however in a traditional security chip one might expect protective shields, plus power smoothing and noise insertion to counter power analysis, and sensors/detectors to counter fault attacks. Some protection may arise from the application and OS software e.g., randomised/repeated operation and dummy cycles, although this may be optimised for the included algorithms. For a new algorithm running on this chip, we should expect some protection from the hardware, although the final algorithm code will need to improve this, which would likely degrade the performance measured in our experiments.

2) SLE78: The SLE78 is an innovative security controller intended for high security applications. Instead of relying mainly on shields and sensors it uses "Integrity Guard" [25], which exploits dual CPUs working in tandem. The claimed features include:

- Dual CPU implementation for fault detection
- Full CPU, memory, Bus and Cache encryption
- Error detection codes on all memories
- Error codes for cache protection
- Address and data scrambling of memories
- Side-channel leakage suppression
- Active Shield

Running the algorithm on the SLE78 offers a good deal of hardware protection with less reliance on added software countermeasures; so we would anticipate less performance degradation when compared with the SLE77.

IV. SOFTWARE DEVELOPMENT

The starting point for the smart card software development was the example code published in 3GPP TS 35.231 [6]. This went through several versions during the project, based on results/feedback and on-going optimisation work. The final versions should be regarded as optimised to the extent that was possible with a generic implementation avoiding chip specific enhancements. Referring to Table I the primary template for the smart card experiments was the generic speed optimised

 ISBN: 978-1-61208-451-0

version 1 that could be built for 8, 16, 32 and 64 bits, and made use of generic loops and macros. The 64 bit option was discounted as being unrepresentative of current smart cards and because legacy C compilers cannot easily cope with integer variables beyond 32 bits. Some minor modifications were made to the initial smart card code, but largely it remained true to the original generic code. Later, in order to understand performance issues relating to the algorithm running on the MULTOS platform, a 32-bit version of the code was part-optimised, which involved expanding the Macros and unrolling the inner loops within the main Keccak functions. The final MULTOS version also used fixed pointers for buffer manipulation. Note that in all versions of the code, the calculation of TOP_c was removed from each function. Within a smart card, this value would be pre-calculated and loaded into protected memory and so there is no need to recalculate it; and doing so could halve a TUAK function's speed.

A. Software Functional Testing

To test TUAK functionality, we used the six test data sets published in 3GPP TS 35.232 V12.0.1 [20]. The data sets were designed to vary all inputs and internal values, and assure correctness of an implementation; they thus also serve well for performance tests. To simplify testing the test data sets were included within the card application. This added an extra static data requirement, but meant that tests could be run by simply specifying the test set within the card test command, or by supplementing the test set with command data. Each command had an execution count so the targeted function could be run from 0 to 255 times (on the same input data). Typically the count would be '1', although '0' was useful for estimating round trip delays and higher counts improved measurement precision.

V. RESULTS

In this section, we present the experimental results, based on the 3GPP test data. The results were obtained via a scripting tool that would send a command message to the card in the form of an Application Protocol Data Unit (APDU) and then time the response. Although card processing time should be consistent and repeatable, scripting tools have tolerances. To compensate, the test commands instruct the card to execute a function multiple times before returning a result. A calibration was also carried out using a protocol analyser.

A. PC Results

The initial performance experiments used to refine the public example code were PC based, with results (in clock cycles) from the various versions (see Table I) summarised in Table II. Note that the cycle number includes pre, post data processing and overheads for a single run of Keccak-1600 (24 rounds).

Variation between minimum and average results arises from the OS. The minimum values are representative of the CPU capability. Generally, speed increased with the target build size.

B. Smart Card Performance

Native card performance was mainly measured on the SLE77; only the 32-bit algorithm was run on the SEL78. The MULTOS results used both chip types for all tests. The results are shown in Tables III and IV.

TABLE II. PC VERSION PERFORMANCE COMPARISON

Versions	Minimum Cycles (average cycles)		
	8-bit	16-bit	32-bit
0 (size opt)	168652(380066)	85988(215250)	
1 (speed opt)	49688(116200)	22496(55343)	7152(9024)
2 (N/A)			
3 (original)	202140(221564)		87350(193371)
4 (N/A)			
5 (unrolled)			6368(10391)
6 (bit-interl)	73120(185217)	59307(131112)	
7 (bit-interl opt)			10216(25570)

TABLE III. NATIVE MODE PERFORMANCE (ms)

Test	Mode/Chip	SLE77			SLE78		
Data		f1f1s	f2345	f5s	f1f1s	f2345	f5s
	8-bit	18.11	18.17	18.11			
1	16-bit	15.17	15.23	15.17			
	32-bit	19.58	19.64	19.51	19.58	19.70	19.51
	8-bit	18.17	18.17	18.17			
2	16-bit	15.23	15.23	15.17			
	32-bit	19.64	19.76	19.58	19.64	19.82	19.58
	8-bit	18.23	18.17	18.17			
3	16-bit	15.23	15.29	15.17			
	32-bit	19.70	19.82	19.58	19.70	19.88	19.58
	8-bit	18.17	18.23	18.17			
4	16-bit	15.17	15.23	15.17			
	32-bit	19.58	19.76	19.45	19.58	19.76	19.51
	8-bit	18.17	18.23	18.17			
5	16-bit	15.17	15.36	15.17			
	32-bit	19.58	20.01	19.58	19.58	20.00	19.58
	8-bit	36.22	36.27	36.19			
6	16-bit	30.16	30.28	30.10			
	32-bit	38.85	39.15	38.67	38.79	39.15	38.60

Normally, when the MULTOS organisation specifies a new function for the Virtual Machine (VM) it would be coded in low-level software and invoked from an Application Programming Interface (API). The API performance should be closer to that of Table III; however as this is currently not the case, the Table IV figures apply. All versions of the application benefit from a typical memory optimisation i.e., the Keccak main buffer (INOUT) was forced into a reserved section of RAM. Using non-volatile memory (NVM) instead made the 8-bit and 16-bit versions three times slower and the 32-bit version five times slower. The "32x" rows represent the

TABLE IV. MULTOS PERFORMANCE (ms)

Test	Mode/Chip	ML4 = SLE77			ML3 = SLE78		
Data		f1f1s	f2345	f5s	f1f1s	f2345	f5s
	8-bit	19882	19952	19796	23837	23947	23962
	16-bit	10749	10826	10702	12824	12917	12838
1	32-bit	6396	6505	6350	7239	7348	7192
	32x	3104	3214	3073	3432	3557	3400
	32p	1529	1575	1529	1623	1654	1622
	32-bit	6474	6568	6396	7332	7441	7254
2	32x	3198	3276	3120	3526	3619	3463
	32p	1544	1576	1529	1638	1669	1623
	32-bit	6537	6615	6396	7379	7504	7254
3	32x	3245	3339	3120	3603	3681	3463
	32p	1560	1592	1529	1654	1670	1623
	32-bit	6427	6552	6349	7269	7410	7191
4	32x	3151	3261	3089	3478	3603	3401
	32p	1544	1591	1529	1623	1669	1622
	32-bit	6443	6708	6412	7301	7597	7254
5	32x	3166	3432	3120	3494	3791	3463
	32p	1544	1622	1529	1638	1700	1622
	32-bit	12543	12808	12402	14211	14492	14071
6	32x	5990	6224	5866	6614	6879	6474
	32p	2980	3057	2949	3135	3198	3105

 ISBN: 978-1-61208-451-0

TABLE V. TEST DATA PARAMETER SIZES

Test Data	K	MAC	RES	CK	IK	Keccak Iterations
1	128	64	32	128	128	1
2	256	128	64	128	128	1
3	256	256	64	128	256	1
4	128	128	128	128	128	1
5	256	64	256	256	128	1
6	256	256	256	256	256	2

"unrolled" version of Keccak, which is a removal of inner loops and macros in the C code, and the "32p" version also uses fixed pointers rather than array index calculations. The smart card test results are further described and analysed in Section VI.

VI. Analysis of the Smart Card Results

To consider the experimental results, it is necessary to be aware of the parameter sizes (bits) inherent in the standardised test-sets, which are summarised in Table V. The test data parameters are designed to exercise TUAK in representative modes of use. Note that for the first five test sets (single iteration) the Keccak core has very similar execution time, with TUAK variations arising from the differing amounts of data to absorb or squeeze out of the sponge (working buffer).

Note that the common/fixed parameters sizes (bits) for the TUAK algorithm are: RAND = 128, SQN = 48, AK = 48, AMF = 16.

A. Performance Target

We need to define an appropriate performance target, so we can start by recalling the target used for the MILENAGE design [4].

> ..."The functions *f1-f5* and *f1** shall be designed so that they can be implemented on an IC card equipped with an 8-bit microprocessor running at 3.25 MHz with 8 kbyte ROM and 300byte RAM and produce AK, XMAC-A, RES, CK and IK in less than 500 ms execution time."...

Technology has advanced since this target was created and it might be difficult to find a SIM chip with these minimal capabilities, and indeed many do not have ROM. Furthermore, the target is ambiguous and could be interpreted that if you ran the functions in sequence each could take 500ms. It is also unclear how much of the ROM and RAM can be used. A more appropriate and modern target was defined during the study.

> ..."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 8kbytes non-volatile-memory (NVM), reserving no more than 300bytes of RAM and producing AK, XMAC-A, RES, CK and IK in less than 500 ms total execution time."...

This revised target definition has been proposed to 3GPP for inclusion in future versions of the standard documents.

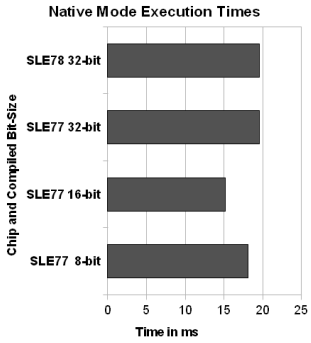

Figure 3. Comparison of Native Mode Execution Times

B. Native Mode

If we consider the results from the native implementation on the SLE77, the function execution times for the various test data sets are quite similar with the exception of test set 6. The latter uses a double iteration of Keccak, which roughly doubles the execution time. As can be seen from Figure 3, compiling the generic code for the different target bit widths affects the execution time, but not by an enormous margin. The most efficient version is the 16-bit target, which provides the best fit for the underlying processor.

Due to practical constraints we only have SLE78 measurements for the 32-bit target, which show similar speed to the SLE77 (native). The extra security features of the SLE78 seem not to penalise performance although there may be added financial cost. The striking observation is that native mode performance satisfies our target by a very comfortable margin. It is therefore reasonable to conclude that provided the algorithm is custom-coded on a typical (rather than highest performance) SIM chip there is no need for a crypto-coprocessor.

This study focussed more on performance than code-size minimisation, however, all native implementations fitted within our memory targets.

C. Platform Mode

Within the study we only considered the MULTOS platform; although a Java Card would make an interesting comparison. The results here were disappointing, although a significant overhead had been expected due to the operation of the secure Virtual Machine and the MULTOS Execution Language (MEL) [26] abstraction. In practice, the best results were around two orders of magnitude slower than native; see Figure 4. Furthermore, the performance improved with increasing compiled bit-size, which suggests that the compilation and MEL interpretation does not map closely to the underlying CPU size for the processing in TUAK.

On inspection of the generic Keccak function one saw extensive use of macros and loops. To determine if they were causing problems for MULTOS, an "unrolled" 32-bit version of Keccak was created, removing macros and inner loops. The results are in the Table IV rows marked "32x" and in Figure 4, showing a doubling of speed. A further improvement was to adapt the algorithm to use fixed location buffer pointers rather than indexed arrays; and the corresponding "32p" version shows a further speed doubling. However, a single function still takes around 1.5s.

 ISBN: 978-1-61208-451-0

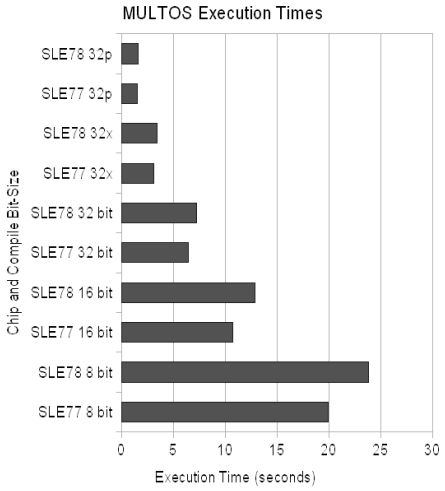

Figure 4. MULTOS f1() Execution Times

If we consider the unrolled Keccak there are many shifts on array contents, however MEL does not have a core shift instruction, but uses shift primitives. The unrolled Keccak is twice as fast as the generic version, partly due to the way that MULTOS handles shifts. The amount of shift on a buffer m can be known at compile time or run time, as shown below.

```
m = m << 3    or    m = m << n
```

The first example is handled as a single use of the shift primitive, whereas the second will loop n times shifting 1 bit at a time. This still leaves a big question mark over the efficiency of the primitive itself (and other bitwise operations).

If we consider the x2 speed-up from pre-computing TOP_c, the x2 from removing loops/macros and the x2 from using pointers, the application is x8 faster than the generic version. However the conclusion is still that the algorithm cannot meet the target performance if loaded as an application on a card platform (MULTOS at least). This suggets it is not practical to add the algorithm to deployed or existing stock cards. To use a card platform, an API would need to be added so that an efficient native implementation could be called.

VII. Security Defences and Performance

Modern SIM cards are normally based on tamper-resistant secure microcontrollers, which inherently have a range of defences against physical, side-channel and fault attacks. Therefore, a TUAK implementation on a SIM platform should be much better protected than an implementation on a general purpose microcontroller, with the latter incurring significant performance overhead to achieve modest attack resistance. If we consider the chips used in our tests then the SLE78 would be expected to offer significant protection against physical, side channel and fault attacks [24] due to the innovative underlying hardware; requiring less software countermeasures (and performance degradation) than a conventional secure microcontroller. The SLE77 would also offer hardware based protection, particularly against physical and fault attacks, but adequately preventing side-channel leakage will require additional measures in software. Fortunately, the SLE77 is quite fast and even if the performance was degraded by an order of magnitude, we could still run *f1, f2345* and *f5s* and meet the overall performance target. MULTOS platforms are known and marketed for their high security and had they been fast enough they would have been expected to offer added OS security to compliment the underlying chip hardware. However, the current view is that a new MULTOS primitive will be needed for the algorithm and so the issues are similar to the SLE77/78.

A. Fault Attack Defences and Performance Impact

The faults used in attacks are normally achieved by voltage glitches, radiation pulses and operating the target device beyond tolerance. The hardware sensors in tamper-resistant smart cards are intended to detect the likely means of fault insertion and prevent a response useful to the attacker; so there is no significant added overhead for the software. A very sophisticated and skillful attack might bypass the sensors, however by adopting TUAK as an openly published algorithm, with diversified card keys, we are avoiding proprietary secret algorithms that might motivate such effort. An added countermeasure could be to run the algorithm twice and only output a response if the results agree; this would counter attacks that analyse correct and faulty results from algorithms. The added countermeasure is probably unnecessary for the chips considered in this work, although halving the speed of operation would still keep it well within specification. Note that an attacker will seek to insert a fault at the most opportune moment, which may be determined from side-channel leakage.

B. Side-Channel Attack Defences and Performance Impact

Timing leakage attacks [27] can be possible when there are observable data dependent delays in the application; in which case added redundancy is needed in the implementation. Timing variations can be sufficiently large that they can be detected despite low level measures to disguise side-channel leakage that might be subject to power analysis. The leakage generation principle is quite simple, e.g., if a variable is true do something time-consuming else do something quick. The variable could represent a value that is tested at the application layer, or just a low-level bit test. A brief inspection of Keccak does not show obvious high-level timing leakage, as there are no conditional branches in the code. However, there could be lower level leakage if bit rotates are used. For example a processor may effect a rotate by shifting the contents of a register up one place and then testing the value that falls out of the register. If the value is '1' then this has to be added back in as the LSB, so unless the designer adds dummy operations, processing a '1' is going to take longer than a '0'.

The Keccak example code has macro names that imply rotate, but on inspection they are buffer shift operations rather than register rotates. However, there could be a timing effects when the compiled target size (8/16/32 bit) does not match the underlying register size. For example if we compile for 16-bits, but the CPU registers are 8-bits then our shift may need to modify the least significant bit of the upper byte based on the bit value shifted out of the lower byte. In the case of native code implementation, developers would be expected to take the CPU size/shift/rotate into account. In the platform approach the mapping between application variables and underlying registers is unclear.

We have assumed that the chips have hardware countermeasures to prevent bit-level side-channel leakage, as software

 ISBN: 978-1-61208-451-0

measures are inferior and significantly impact performance. For example, Hamming-weight equalisation is a technique that seeks to reduce leakage by ensuring that for each bit transition there is a complementary transition; so as a '1'changes to '0' there is also a '0' changing to '1'. In a practical implementation this could for example be a 16-bit processor where the lower 8-bits of a register handle the normal data and the upper 8-bits handle the complementary data. However, at the physical/electrical level, the register bits are unlikely to have equal contribution to the leakage and so Hamming-weight equalisation may not deliver a sufficient reduction. The impact on execution speed is also significant, as it is necessary to clear registers before and after use, and so a ten-fold rather two-fold reduction in performance should be anticipated.

VIII. Conclusions and Future Work

The main conclusion is that it is feasible to implement TUAK in software on typical smart card/SIM chips and meet the performance target for 3G/4G authentication algorithms, without the need for a cryptocoprocessor. Native mode implementation is required and so for a card platform (such as MULTOS) this should be supported via API calls. Processor and memory requirements are very modest suggesting that TUAK could meet performance targets even when implemented on simpler legacy CPUs. Although there is no high-level data dependent timing in TUAK, there is some potential for data dependent side-channel leakage due to shift operations, which will require countermeasures. Whilst high-end smart card chips (like the SLE78) may offer significant hardware-based resistance to side-channel analysis, other chips will require help from software countermeasures. Such measures may significantly impact performance; however the SLE77 results show that function execution time could be reduced by an order of magnitude and still satisfy the performance target. The primary impact of the work is that by showing TUAK to be a practical back-up or alterative to MILENAGE for typical SIM platforms, it will be adopted as a preferred public algorithm (initially in M2M systems); displacing proprietary solutions that are often the target and motivation for attack.

On-going work is considering a less advanced/legacy smart card chip (S3CCE9E4/8), side-channel leakage, and whether TUAK could be re-used in other applications. Preliminary results indicate that TUAK is sufficiently fast for use on more limited chip platforms, and this suggests it might also be a candidate for Internet of Things protocols. In fact re-using a 3G algorithm is not a new idea as MILENAGE has already been reused outside of mobile communications.

An initial collection of power traces from the original SLE77 TUAK implementation, shows the rounds and the round structure, although more traces and detailed analysis are still required to clearly determine data dependent leakage.

References

[1] (2016, Jan.) The European Telecommunications Standards Institute website, [Online]. Available: http://www.etsi.org/

[2] (2016, Jan.) The Third Generation Partnership Project website, [Online]. Available: http://www.3gpp.org/

[3] M. Mouly and M. Pautet, The GSM System for Mobile Communications, Cell and Sys (1992)

[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] 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)

[7] G. Bertoni, J. Daemen, M. Peeters, and G. van Aasche, The KECCAK Reference, version 3.0, 14 (2011)

[8] 3GPP TR 35.934: 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 4: Report on the design and evaluation (2014)

[9] 3GPP TR 35.936: 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 6: Security assessment (2015)

[10] 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)

[11] (2016, Jan.) eBACS: ECRYPT Benchmarking of Cryptographic Systems, [Online]. Available: http://bench.cr.yp.to/results-sha3.html

[12] Y. Jararweh, L. Tawalbeh, H. Tawalbeh, and A. Mohd, Hardware Performance Evaluation of SHA-3 Candidate Algorithms, Journal of Information Security, Vol 3, Number 2, p69-76 (2012)

[13] 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)

[14] (2016, Jan.) MULTOS website, [Online]. Available: http://www.multos.com/

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

[16] F. Hillebrand, GSM and UMTS - The Creation of Global Mobile Communication - Wiley, (2002)

[17] 3GPP, TS 33.102: 3G Security; Security Architecture (1999)

[18] 3GPP, TS 33.401: Telecommunications Specification Group Services and System Aspects; 3GPP System Architecture Evolution (SAE); Security architecture (2012)

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

[20] 3GPP, TS 35.232: 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 2: Implementers' Test Data (2014)

[21] 3GPP TS 35.233: 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 3: Design Conformance Test Data (2014)

[22] Infineon, SLE77CLFX2400P(M) Short Product Overview v11.11, (2012)

[23] Infineon, SLE78CAFX4000P(M) Short Product Overview v11.12, (2012)

[24] K. Mayes, and K. Markantonakis, Smart Cards, Tokens, Security and Applications, Springer (2008)

[25] (2016, Jan.) Infineon, Integrity Guard White Paper, [Online]. Available via: http://www.infineon.com/

[26] MULTOS, Developer's Reference Manual MAO-DOC-TEC-006 v1.49, (2013)

[27] P. Kocher, Timing Attacks on Implemenetationsof Diffie-Hellman, RSA, DSS, and other Systems, Advances in Cryptology CRYPTO 96 Volume 1109 LNCS p104-113 (1996)

Acknowledgment

The authors would like to thank members of ETSI SAGE for their expert advice.