Secure Shared Processing on a Cluster of Trust-Anchors

Keith Mayes

Royal Holloway, University of London, Egham, Surrey, UK,
keith.mayes@rhul.ac.uk

Abstract. Attacks on computer systems and networks have never been more prolific, hence the great effort from government, industry and academia, to identify and adopt information/cyber security best-practices. Most of this effort has been directed to the logical design and operational security of systems, however the security of implementation is also vitally important, especially for critical and machine-to-machine infrastructures. One approach to underpinning implementation security, is to distribute, certified-secure chips, as hardware security modules (HSM), to provide strongly attack-resistant and trusted endpoints for protocols. A risk with physically deploying fixed function HSMs is that they may need to have a long life-time, yet be unable to support new algorithms and protocols in response to evolving threats and defenses; so a manageable secure platform is attractive. Existing single-chip platforms have specialist hardware security, including crypto coprocessors to help performance, however their general processing is slow, due to the secure platform software defenses, within what are small, low-cost and low-power chips. In this research we explore the idea of multiple HSMs sharing resources on security processing tasks, without compromising that security via inter-HSM communications. The proposal and related performance experiments center around clusters of up to eight HSMs, using a communications protocol, based on Offset Codebook authenticated encryption; sharing resources for processor intensive tasks. A localised cluster of MULTOS Trust-Anchor chips was used for experimentation, although the principles of the proposal extend to clusters that are widely dispersed.

Keywords: Authenticated Encryption, OCB, MULTOS, HSM, Security, Cluster

1 Introduction

Deployed Hardware Security Modules (HSM) are seen as a way of protecting critical infrastructure and Machine-to-Machine (M2M) systems against implementation attacks. The HSMs may be Security Chips (SC), which have undergone formal Common Criteria (CC) [2] evaluation to a high level, including independent laboratory testing of tamper and attack resistance. An SC would include a Crypto-Coprocessor (CCoP) for efficient support of common cryptographic algorithms and related processes, but otherwise the chips are low-power, low-cost and with limited CPUs and memory. Direct implementation of fixed-functionality on the processor, is best for performance, but for widespread and long-term deployment, a manageable secure platform is required, which is abstracted from the underlying hardware. In this research we use the

MULTOS [11] platform based on its high levels of CC, which has lead to its use in payment cards and passports; in particular we use the MULTOS Trust-Anchor [12] SC, which is intended for Internet of Things (IoT) security applications. To illustrate the performance challenge of using such a small secure platform for complex security processing, we refer to a previous study [10], which showed that for an identical chip, the performance difference between non-secure native code and highly secure MULTOS application code, could be two orders of magnitude; with secured native code, expected somewhere in between. In this research, we explore the idea of a cluster of SCs working collaboratively on a security process to overcome performance restrictions. This presents a major security challenge, as the evaluated SC relies heavily, on security processing being carried out within the single chip, using secret or private keys which also do not leave the chip, and with no intermediate results ”leaking” from the chip. The distribution of long-term keys and cryptographic credentials is not a primary concern, as this is inherently supported within MULTOS, however SC collaborative working suggests that sensitive data and intermediate results will be exposed on communications interfaces. The CCoP supports secret key algorithms with relative efficiency, so one could imagine using AES [4] encryption to preserve confidentiality of communications, however this alone is insufficient, as a man-in-the-middle attack may modify or insert messages to alter the process result. The chosen approach was to design a protocol around Authenticated Encryption (AE) [5], which has been studied [10] with respect to MULTOS (and native) platforms for the EMV payment card standards at the request of EMVCo [3]. There are several AE modes which could be used, however the earlier work showed that for messages of 40bytes or more, Offset Codebook (OCB) [8] was the most efficient mode for the MULTOS platform; and chosen for this research, based on anticipated test message sizes. The OCB AE implementation made use of 128-bit key AES as the block cipher. To investigate the proposed multi-SC approach, we created a generic load case, whereby a processing task of controllable duration could be split into eight parts and then shared between the available supporting SCs. For the experimental system, we chose the MULTOS Trust-Anchor (TA) as the SC type; the TA being intended for IoT security applications. To evaluate the performance aspects, a cluster of eight TAs, was used under the supervisor of a Master Trust-Anchor (MA).

This report first introduces the MULTOS Trust-Anchor and the experimental cluster in Section 2 and then provides goals and requirements for a security protocol in Section 3. A proposal for an AE-based protocol is described in Section 4, and experimental results are presented and discussed in Section 5. Conclusions and suggestions for future work are presented in Section 6.

2 MULTOS and Trust-Anchor

MULTOS is a robust and highly secure Operating System (OS), backed by a large industry consortium [11]. It has an on-chip Virtual Machine (VM) that was originally used to provide a secure multi-application platform for smart cards used in high-security applications, such as payment and passports. MULTOS devices have CCoP support and use different memories for code, non-volatile data, and RAM for the stack and variables.

The memory map is shown in Figure 1, but note that the physical RAM space in particular, is usually just a fraction of what could be theoretically addressed. A critical aspect of MULTOS devices is the high-levels of CC evaluated security, which includes extensive independent lab-testing of resistance to attacks, including physical, side-channel [6] [7] and fault [1]. Another vitally important aspect of MULTOS, is the ability to strictly control the data and applications to be loaded into the chip; which is enforced by a Public Key Infrastructure (PKI), in which developers and their proposed load-units have to be registered and certified. The loading security is well suited to device personalisation and loading post-deployment. Source code development for MULTOS is normally in the *C* language, which is compiled to MULTOS Execution Language (MEL), the VM is 16-bit, with 32-bit extensions for memory access; an introduction can be found in [9]. MULTOS chips intended for smart cards are not suited to our cooperative cluster proposal, as they have limited I/O and are not intended to be free-running processors; the MULTOS Trust-Anchor does not have these limitations, and offers more RAM memory.

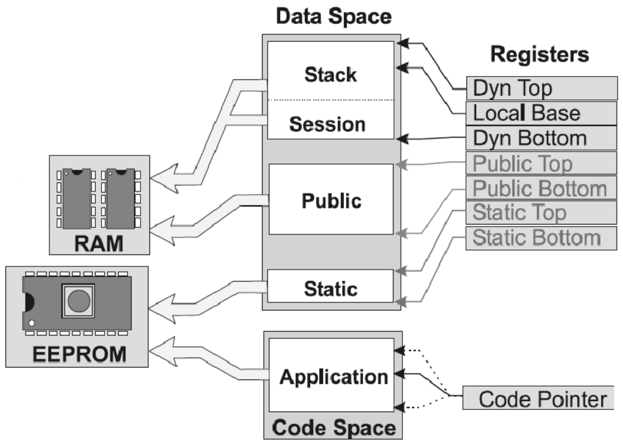

Fig. 1. MULTOS Application Memory Architecture

2.1 Trust-Anchor

The MULTOS Trust-Anchor (TA) device [12] is intended to offer highly attack-resistant security, for IoT applications. It offers 9.5kbytes of stack and session RAM and a further 3.5kbytes of public RAM, and a total of 315kbytes Non-Volatile Memory (NVM) for the secure OS, application code and static data. It has a sensor and shield protected dual processor CPU, a 2kbits+ CCoP, internal timers and random number generation, a legacy smart card interface, a USB interface and up to 12 GPIO lines. Two of the GPIO lines can be configured to form an I2C bus; which is normally used for the control of peripherals, however, in our experiments, it was re-purposed to connect the cluster of devices. The actual chip package area is only 25mm ^{2}, although larger breakout boards were used for development.

2.2 The Cluster

Our proposal makes use of a cluster of TA security devices as shown in Figure 2. The cluster has up to eight TAs acting as secure processing service providers (SPSP) to the MA; which itself acts as a SPSP to the requester, represented by the *PC* in Figure 2. In normal secure use, each device would be personalised with a unique address and security credentials (including cryptographic keys), however for experimentation, the TAs determine their (hard-coded) address from reading three of their GPIO lines. The address is used to select from a set of stored personalisation profiles, allowing a common build and configuration to be used during performance testing. Note that the very limited RAM mentioned in 2.1, imposes restrictions not just on the TA applications, but also on the communications buffering, which dictates some design choices in multi-TA cluster processing.

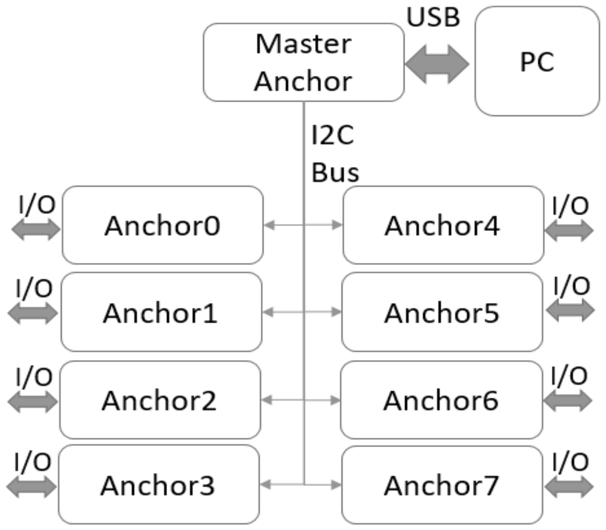

Fig. 2. Experimental Cluster of Trust-Anchors

3 Security Goals and Requirements

The primary security goal within our research, is that the use of a TA cluster for security-sensitive processing, is no less secure than carrying out the entire process within a single chip. The protection of a centralized authentication entity and server is out of scope for this study, but is expected to follow security best-practices, suitable for it to be a trusted endpoint. We assume that the TAs within our cluster and the communications between them are accessible to attackers, however as the MULTOS TA is very strongly attack-resistant, (as can be proven by CC evaluation), we do not focus on requirements for TA implementation attack protection, but rather on the inter-TA communications.

3.1 Inter-TA Communication Attacks

The difficulty of compromising a communications interface or path, depends on the design and technology used, but also on its location and accessibility to the attacker. In our proposal, we assume that attackers can both passively monitor transmissions and actively attempt to modify transmissions, or insert fake transmissions (often referred to as Man-In-The-Middle attacks). Therefore, to protect sensitive communications between two legitimate parties (the TAs), requires the following defenses:

1. Prevention of security information discovery by passive eavesdropping
2. Detection and rejection of fake messages from non-authenticated sources
3. Detection an rejection of replayed, or re-ordered legitimate messages
4. Detection and rejection of modified legitimate messages
5. Detection of blocked and missing legitimate messages
6. Detection of Denial of Service (DoS) transmissions

Eavesdropping within the planned context, could potentially expose sensitive data, cryptographic keys and credentials, partial algorithm results, or information that could be exploited in other attacks. The solution is to encrypt transmissions using a best-practice algorithm, such as AES. A protocol that incorporates source authentication will detect fake message insertion, and cryptographic integrity protection will detect modification of legitimate messages. Replayed and re-ordered messages are detected by protocols that incorporate a count, or initialization value, which updates for each protected message block; a mechanism that can also detect missing messages in a sequence. With the exception of DoS our security requirements can be supported on security chips by using Authenticated Encryption (AE) protocols, standardised in ISO/IEC 19772 [5]. A DoS attack is detectable by unexpected transmissions and failures, and in the planned context, the MA has the option to return to non-clustered processing, at the expense of performance. In the next section we propose a protocol that leverages from best-practice standards, satisfies the security requirements and will permit practical performance evaluation.

4 Secure Cluster Protocol Proposal

Our proposal assumes that each TA in the cluster has been securely personalized with unique long-term cryptographic credentials such as keys and IDs; so they are uniquely addressable and can support the operational and management basis of a security end-point. We also assume that the desired functionality used in cluster-based processing has been pre-loaded into the devices, which can be verified via the TA's Service Table, which is reported to the MA. All personalization is achieved using the well-proven MULTOS management protocols. Only the MA is aware of the overall cluster operation and associated data, other TAs just respond to service processing requests, based on sub-sets of data. The only common key that is personalized to the sub-set of devices that form the cluster, is the default cluster key. This key is only used to generate a new cluster session key, when requested by the MA.

The MA is responsible for the cluster (and associated session key) establishment as well as management of cluster processing. In order to do this securely, it needs a method of protecting transmissions to maintain confidentiality and integrity of communications between authentic parties. Authenticated Encryption (AE) is a solution for the security protection problem and having previously [10] been proven to be practical on MULTOS platforms, was selected as the method for this research. Various modes of AE have been standardized ISO/IEC 19772 [5]. It has been shown in [10] that Encrypt-Then-MAC (ETM) mode is efficient for very small data payloads (up to 32 bytes); but for larger messages, OCB is faster. The latter was chosen for this work, and introduced below.

4.1 Offset Codebook Authenticated Encryption

The Offset Codebook approach to AE was pioneered by Phil Rogaway [13], and also described in RFC 7253 [8]. It was included as *mechanism 1* in ISO/IEC 19772 [5]. The mechanism can be briefly described with reference to Figure 3. As with most AE mechanisms the plaintext message is split into blocks ready for the block-cipher operations, however, OCB differs in that an incomplete last block does not require padding. The example of Figure 3, has three complete and one incomplete message blocks (M1-3, M*), producing an output sequence of C1-3, C* plus an extra tag T. Note that there is additional processing (of some significance) in computing the initialization vector, although this is only recommended once every 64 blocks. We do not use Authenticated (but not encrypted) Data, so *Auth* can be disregarded in the diagram. There are variants of OCB; we used OCB2, as the most efficient mode found in [10].

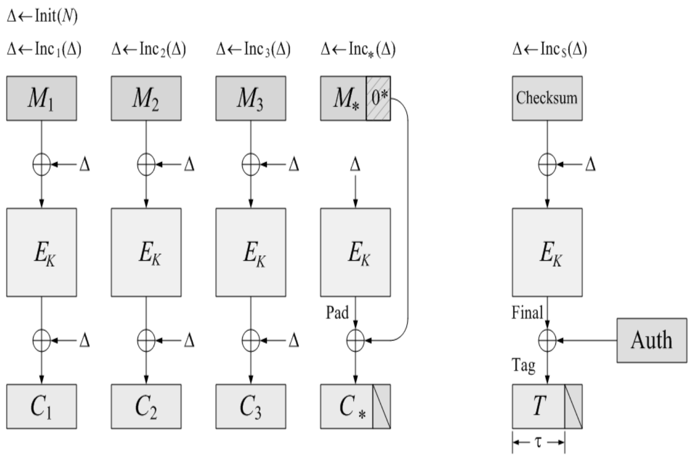

Fig. 3. OCB with Incomplete Blocks [Rogaway]

4.2 Cluster and Session Key Establishment

HSMs are typically personalized before use, which involves the secure loading of unique long-term cryptographic keys and related credentials, as well as common functions and data. It is best-practice to use open standardized cryptographic algorithms and diversified keys and related data, so that HSMs are unique, and in the unlikely event of compromise, will not aid an attack against other devices. In practice, long-term keys are normally used to generate session keys that are used operationally. We follow these principles in our Trust-Anchor proposal, except that the SCs able to become part of the cluster, have also been personalized with a common cluster key, used to establish cluster session keys, for use between the TAs and MA. The protocol steps for establishing the cluster session key are shown in Figure 4, with symbols in Table 1; noting that the figure represents the interaction with a single selected TA, whereas in practice the MA may interleave its interaction across multiple TAs, as best suits parallelism and efficiency.

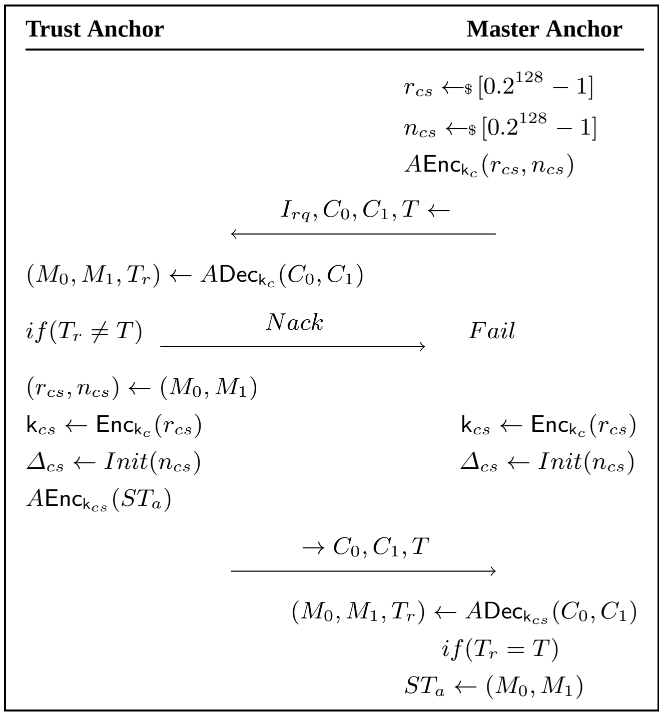
Trust Anchor
Master Anchor
$r_{cs} \leftarrow_{\$} [0.2^{128} - 1]$
$n_{cs} \leftarrow_{\$} [0.2^{128} - 1]$
$A\mathsf{Enc}_{\mathsf{k}_c}(r_{cs}, n_{cs})$
$I_{rq}, C_0, C_1, T \leftarrow$
$(M_0, M_1, T_r) \leftarrow A\mathsf{Dec}_{\mathsf{k}_c}(C_0, C_1)$
$if(T_r \neq T)$
$Nack$
$Fail$
$(r_{cs}, n_{cs}) \leftarrow (M_0, M_1)$
$\mathsf{k}_{cs} \leftarrow \mathsf{Enc}_{\mathsf{k}_c}(r_{cs})$
$\mathsf{k}_{cs} \leftarrow \mathsf{Enc}_{\mathsf{k}_c}(r_{cs})$
$\Delta_{cs} \leftarrow Init(n_{cs})$
$\Delta_{cs} \leftarrow Init(n_{cs})$
$A\mathsf{Enc}_{\mathsf{k}_{cs}}(ST_a)$
$\rightarrow C_0, C_1, T$
$(M_0, M_1, T_r) \leftarrow A\mathsf{Dec}_{\mathsf{k}_{cs}}(C_0, C_1)$
$if(T_r = T)$
$ST_a \leftarrow (M_0, M_1)$

Fig. 4. Establishing Cluster Membership and Session Values

Referring to Figure 4, the MA, starts by generating a random number and a random nonce, and copying into two AES blocks, which are AE encrypted using the long-term cluster secret key and its associated *default* Δ_c. The cipher blocks and token are then sent to the TA, which AE decrypts the message into two plaintext blocks and regenerates the token. If there is a mismatch between local and received tokens, a *Nack* response indicates the protocol has failed, otherwise, the random number and nonce are copied from the plaintext, and the random number is AES encrypted under the cluster key to generate the cluster session key. The TA then AE encrypts its Service Table (ST) data, under the session key, and sends to the MA (which also generates the key). The MA, AE decrypts the message using the session key, and if the token is verified, the cluster membership and session key establishment has succeeded for the particular TA. The ST

Table 1. Symbol Definitions

Symbol	Description
k_c	Personalized long-term AE cluster key (128-bit)
k_{cs}	The current cluster session AE key (128-bit)
r_{cs}	Random for cluster session key generation (128-bit)
n_{cs}	The current AE cluster session nonce (128-bit)
Δ_c	Initialization value for long-term cluster key
Δ_{cs}	Initialization value for session cluster key
M_j, C_j	The jth message and cipher blocks (128-bit)
T_i, T_r	The ith AE token, and recomputed token (64-bit)
$A\mathsf{Enc}_{\mathsf{k}_y}$	Authenticated Encryption under key set y
$A\mathsf{Dec}_{\mathsf{k}_y}$	Authenticated Decryption under key set y
$\mathsf{Enc}_{\mathsf{k}_y}$	AES Encryption under key y
$Init()$	Initialization value generator
$Func_x()$	Service Function for Request x
ST_a	Service Table for Anchor a
DI, DO	Service Input and Output Data
I_{rq}, S_{rq}, R_{rq}	Initial, Service and Read Requests
$Flag_a$	Service Completion Flag for Anchor a

data informs the MA of the services that the TA can support in the cluster operational phase. The establishment process may be repeated, to refresh session keys, nonces, and Δ_{cs}. The session key establishment has messages with 32-bytes of data, which could have efficiently been protected by the ETM AE mode, however, operational messages are larger and will dominate performance, hence the use of OCB for all transmissions.

4.3 Operating the Cluster

Referring to Figure 5 and Table 1, we see how the MA requests assistance with a security-sensitive data processing task. From the cluster set-up phase, the MA learns the Service Table of the TAs, so only requests supported services. The MA generates a service request, by AE encrypting, under the cluster session key, the input data (DI) to be processed, and then sends this along with the token and service type request to the TA, which AE decrypts the input data with the cluster session key. If the recomputed and sent tokens do not match then a *Nack* is sent to the MA to indicate failure. If the tokens match then the TA immediately sends an *Ack* to the MA, indicating that the request has been accepted and the service function will commence. The MA then waits (actually works with other TAs or processes itself) until a completion flag is detected, upon which it sends a read request for the output data (DO). The TA responds with the DO, AE encrypted under the session key, and the corresponding token. The MA AE decrypts the message and if the tokens match, accepts the DO. Note that completion flag detection can be achieved by either a GPIO signal to the MA from the TA, the MA polling the status of the TA, or a timer expiry. Optimum task segmentation and scheduling would minimize the idle time of the MA and TAs.

Refreshing and Synchronizing To successfully AE encrypt and decrypt an OCB transmission sequence between MA and TA, requires a shared cluster session key, but also the correct initialization vector. The vector is not static and is incremented after each block encryption, so there is potential for loss of synchronism, either due to natural, or malicious message disruption. As the MA knows exactly how much DI has been sent and how much DO is expected for a particular TA, it can, estimate and try alternative vectors based on lost message scenarios. If this is unsuccessful then the MA can re-establish synchronism, by repeating the cluster session key establishment (for all TAs) using fresh nonce and random values. With OCB, it is best-practice to re-fresh the initialization vector after the protection of 64 x 16-byte message blocks.

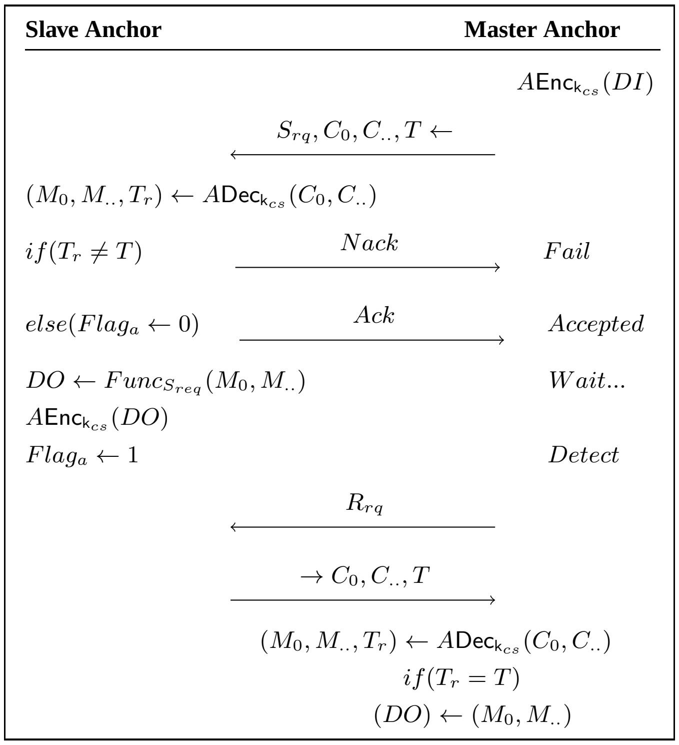
Slave Anchor
Master Anchor
$A\mathsf{Enc}_{\mathsf{k}_{cs}}(DI)$
$S_{rq}, C_0, C_{..}, T \leftarrow$
$(M_0, M_{..}, T_r) \leftarrow A\mathsf{Dec}_{\mathsf{k}_{cs}}(C_0, C_{..})$
$if(T_r \neq T)$
$Nack$
$Fail$
$else(Flag_a \leftarrow 0)$
Ack
$Accepted$
$DO \leftarrow Func_{S_{req}}(M_0, M_{..})$
$Wait...$
$A\mathsf{Enc}_{\mathsf{k}_{cs}}(DO)$
$Flag_a \leftarrow 1$
$Detect$
R_{rq}
$\rightarrow C_0, C_{..}, T$
$(M_0, M_{..}, T_r) \leftarrow A\mathsf{Dec}_{\mathsf{k}_{cs}}(C_0, C_{..})$
$if(T_r = T)$
$(DO) \leftarrow (M_0, M_{..})$

Fig. 5. Service Request and Response

4.4 Experimental Use-Case

The experimental test cluster, consists of eight TAs and the MA, connected via an I2C bus. Each TA has an internal flag bit used to indicate *done* when a task has been completed. The flag state is is duplicated on one of the TA's GPIO (output) pins, which is connected to a MA GPIO pin (input). This provides the MA with the option to determine TA processing state without raising queries over the I2C bus, which could affect performance tests; and also provides external means to record and visualize the effectiveness of the TA scheduling. To permit a common software build for testing, each TA has three GPIO inputs to indicate its I2C address, whereas the address of the MA is implicit. The MA is connected to a PC via a USB port; with the PC providing control, performance measurement and timing logic analysis. Each TA makes use of in-built millisecond timers to simulate processing tasks of varying difficulty and duration; as requested by the MA.

Initial Test and Modelling A TA, in common with most SCs, has very limited RAM and so buffering for preparation and transmission of message data also has to be limited. For testing, we assume that the MA can support a maximum of 1kybte buffering for data, either waiting to be processed or in transmission, so 512-bytes for data needing cluster processing and another 512-bytes for the result. For our eight cluster experiments we break this data into eight sub-sections of 64bytes each; with a delegated processing task, processing this amount of input and returning the same amount of output. Therefore, all our AE payloads will be 64 bytes of encrypted data (equivalent to four AES blocks) plus the authentication token (eight bytes). There is a clear overhead in outsourcing security processing tasks, this includes the times for MA data-input AE encryption, TA data-input AE decryption, TA processing, TA result AE encryption and MA result AE decryption. The TA processing contribution must be sufficiently advantageous to justify the four cryptographic processes. The MA also needs to efficiently schedule the TAs, to minimize their wait time, both to start processing and to deliver their results. The 8-cluster schedule is easiest to visualize, as shown in Figure 6, with time progressing from left to right. MA AE encryption and AE decryption are represented as MAE and MAD, and similarly TAE and TAD for the TA, with the longer bars representing TA data processing.

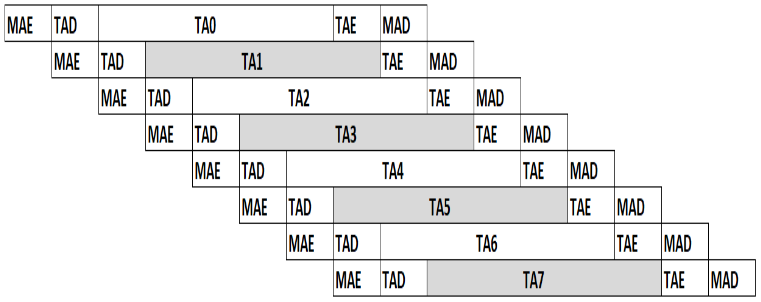

Fig. 6. Optimal 8-Cluster Schedule

The displayed schedule is optimal in that the MA encrypts eight input data messages and decrypts eight result messages without waiting or being idle. Whether it is practically viable, is critically dependent on the duration of the AE cryptographic operations relative to the duration of the overall processing task. The first part of the experiment was therefore to determining OCB AE encryption and decryption times for 64byte data payloads. It was tested by using the PC to send fixed test messages to the MA, running the process multiple times to improve accuracy, and timing the response using internal timers, to avoid communications aspects. Communications delays are considered in later sections when the I2C bus is used.

5 Experimental Results

The experiments were split into two stages. The Initial stage benchmarked the OCB AE performance on the TA, which enabled the best-case multi-TA cluster performance, to be accurately modelled for the TA. When normalized by the AE encryption/decryption time, the results become relevant to other processing platforms. The initial results did not consider transmission delays (as these are channel dependent), or those due to imperfect scheduling (implementation dependent); these aspects were addressed in the transmission and scheduling results stage.

Initial Results The first results are shown in Table 2, indicating the TA speed for OCB2. Note that the encryption and decryption processes are the same (just an added token check for decryption), so the results represent both processes. For interest, a range of data payloads is considered, although the 64 byte process of 42.09ms is the one that we are interested in. It is worth noting that the performance is a little slower than the results in [10] which were obtained on an older MULTOS smart card. The explanation from MULTOS was that the TA has additional redundancy, to offer further enhanced attack-resistance.

Table 2. Trust-Anchor OCB Process Times (ms)

Data Bytes	16	32	**64**	128	192
Duration (ms)	16.17	24.77	**42.09**	76.65	111.13

Using these results, we are able to model, predict and compare the best possible performance for the single MA and when assisted by clusters of two, four and eight TAs. The absolute values are listed in the top rows of Table 3; and show that as the overall processing time increases with respect to the AE operations, the speed gain follows the number of TAs in the cluster. We can express this generically by normalizing the values with respect to the AE operation time; as shown in the lower rows of the table.

Table 3. Comparison of Cluster Processing Predictions

TAs	Absolute Process Times (ms)								
8	672	672	712	837	962	1087	1212	1337	1462
4	714	714	878	1128	1378	1628	1878	2128	2378
2	714	1047	1462	1962	2462	2962	3462	3962	4462
1	0	1000	2000	3000	4000	5000	6000	7000	8000
	Normalized Process Times (multiples of AE time)								
8	16.0	16.0	17.0	19.9	22.9	25.9	28.9	31.8	34.8
4	17.0	17.0	20.9	26.9	32.8	38.8	44.7	50.7	56.6
2	17.0	24.9	34.8	46.7	58.6	70.5	82.4	94.3	106.2
1	0	23.8	47.6	71.4	95.2	119.0	142.9	166.7	190.5

When the overall process time is still comparable with that of the AE operations, the potential for gain depends on the crossover of the cluster performance curves as illustrated in Figure 7. The single TA case is actually the timing for running the process on the MA, so does not involve AE operations. As our test case consists of eight messages, each with 64bytes of data payload, the minimum processing time for a multi-TA cluster is 16 times the AE duration; the time it takes for the MA to AE encrypt the eight inputs and then decrypt the eight results.

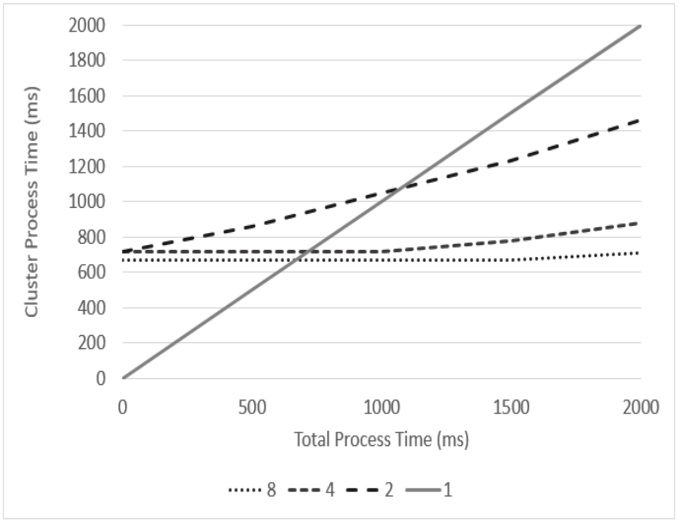

Fig. 7. Cluster Process Times Excluding Communications

Transmission and Scheduling Results The results presented in this section are based on a specific scheduling and communications implementation, of a supporting cluster with either two, four or eight TAs, using the short-range I2C bus for communications and control. We assume optimal scheduling as illustrated in Figure 6, however, we now also assume there is a time delay between the MA deciding to initiate a secure TA task and the TA beginning to AE decrypt the input data; similarly there is a delay between the TA completing AE encryption of the result, and it arriving back at the MA. The control and scheduling of a secure TA process is illustrated in the captured traces of Figure 8, for a dummy 1ms duration task.

The *SCL* and *SDA* traces are the *I2C* bus signals, shared by the MA and TAs, as the primary means of communication. The *TA-Flag* trace is a secondary communications means, in the form of a normally low GPIO output from a TA (input to the MA) that is raised high to indicate that the TA has completed its processing task. The *TA* trace is used to indicate TA processing; first low when the MA data is being AE decrypted (TAD), high for the dummy 1ms task, then low when the result is being AE encrypted, then high again waiting for result collection and its next task. Clearly, the time between the MA starting to transmit AE encrypted data to the TA and the TA starting to AE decrypt, is not insignificant, and neither is the time from the TA completing its AE encryption to the time that the MA has received it to start AE decryption. The bulk of the time lag is due to transmission on the I2C bus, which is running at the default

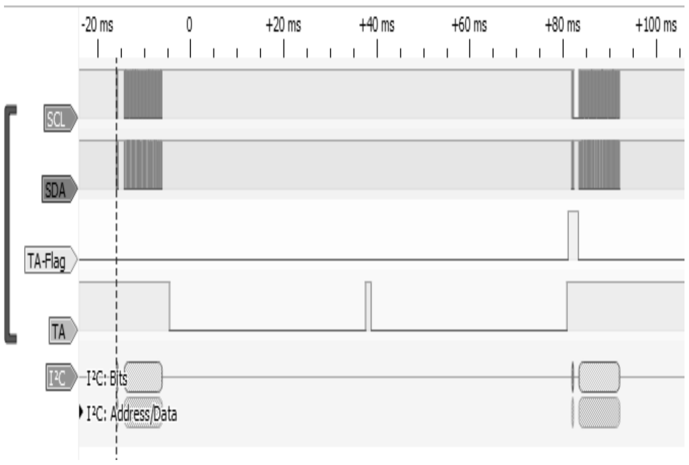

Fig. 8. TA Processing Schedule Trace

100kHz clock speed, with the remainder due to event handling delays within the MA and TA. We investigate the starting lag in more detail, as illustrated in Figure 9

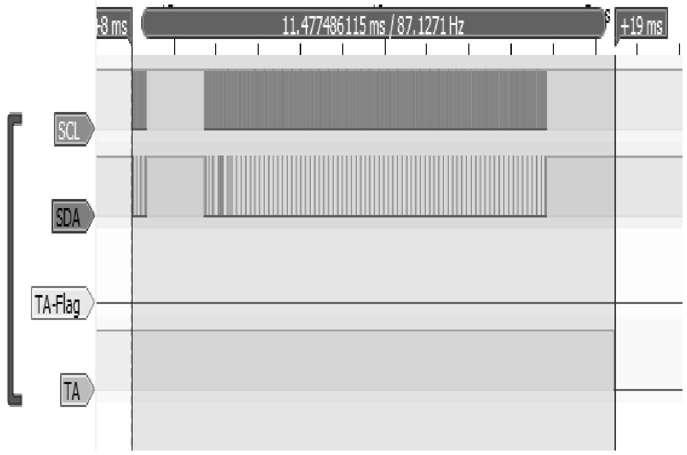

Fig. 9. TA Starting Delay Trace

From the MA beginning transmission of an AE encrypted message, to the TA starting to AE decrypt it, there is a delay of 11.48ms. Now considering the result lag in Figure 10, we see delays between the end of the TA AE encryption and the assertion of its flag, and then to the triggering and completion of communication with the MA. The overall lag at 11.31ms is very similar to the starting lag.

These results can be extended to various conditions by modelling as an extension to the TAE and TAD times, and will increase an effective TA response time by 22.79ms. Because of the pipeline scheduling, the added delays have only minor effect. For an eight, four, and two TA cluster, the processing time is only extended by 22.79, 45.58 and 91.16ms respectively. The final performance graph is shown in Figure 11, with

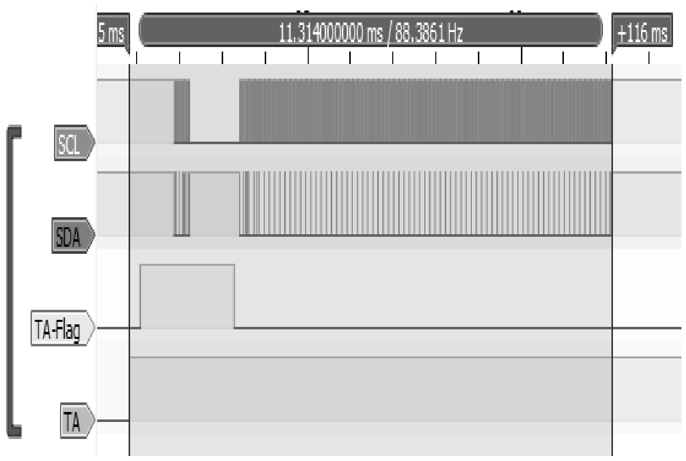

Fig. 10. TA Result Delay Trace

vertical axis normalized to the AE duration, for relevance to the TA, but also to other types of processor. The shape of the graph (up to 2000ms) is very similar to that in Figure 7, and we can see how the gain improves with even longer tasks, shared between the TAs.

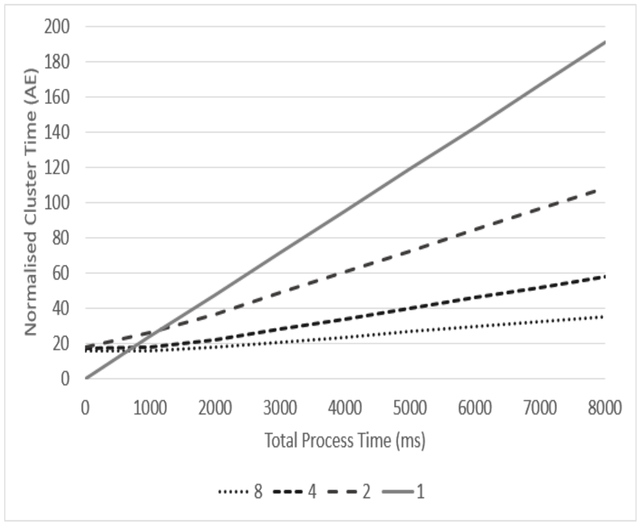

Fig. 11. Cluster Execution Normalized to AE Time

6 Conclusion and Future Work

The study investigated the practicality of secure collaborative processing via a cluster of security chips, in the form of MULTOS Trust-Anchors (TA); whose attack-resistance can be Common Criteria evaluated to a high level. The choice of the MULTOS platform provided justification to focus on inter-platform communications security, rather than chip attacks. Fundamental to high levels of security evaluation, is that for a security sensitive process, partial results cannot leak from the chip, or be manipulated, whereas that is exactly what is at risk with a cluster approach. To overcome this problem, an inter-device secure protocol was proposed based on Authenticated Encryption (AE). This first involved a protocol for session key and initialization vector set-up, using long-term personalized keys; although the main investigative focus was on the subsequent operational protocol, used for sharing and processing the security sensitive tasks. The protocol satisfied the communications security requirements, ensuring the privacy and integrity of transmissions between authenticated parties, based on an OCB standardized mode of AE; which has previously been proven to be practical on MULTOS. The security protocol comes at a cost to performance (compared to a non-secured protocol), which was initially tested and modelled by bench-marking the raw OCB speed on the TA, without the inclusion of transmission and scheduling delays. The main tests considered a processing cluster, with up to eight TAs, working on a task which could be split into eight parts; each with 64 bytes of data input and output. The minimum overall duration under these conditions, was the time for the MA to complete 16 AE operations (672ms), which can be considered as the threshold processing duration, after which the cluster becomes increasingly useful. An investigation into communications and scheduling aspects over the I2C bus, showed delays effecting the start and end of TA processing. However, due to the parallel scheduling, the delays only extended the overall processing time by 22.79, 45.58 and 91.16ms, for clusters of eight, four and two TAs, respectively. The secure cluster protocol has proven practical for the MULTOS TA devices working locally over the I2C bus, and is advantageous for process durations that exceed the threshold processing duration.

6.1 Future Work

Authenticated encryption is designed for general use and is not restricted to localized transmission; it was only the availability and convenience of the I2C bus that led to its experimental use. It would be very interesting to implement an alternative communication channel that would permit a widely dispersed cluster of TAs to collaborate on a secure process. Other secure chips could also be tried, although the TA results normalized by AE time, should allow reasonable prediction of performance, once the AE duration has been bench-marked for the alternative chips.

Acknowledgment

The author would like to thank Chris Torr from MULTOS, for his valuable support and guidance; and Crisp Telecom Limited for use of test and development resources.

References

1. D. Boneh, R. Demillo, and R. Lipton, “On the importance of checking computations,” in *Advances in Cryptography - Eurocrypt 97*, volume 1233, pp. 37-51, Springer Verlag, 2013.
2. CC, ”Common criteria for information technology security evaluation part1: Introduction and general model,” version 3.1 release 4, September 2012.
3. EMVCo, http://www.emvco.com/ [retrieved: May, 2020].
4. FIPS, “Federal Information Processing Standards, Announcing the Advanced Encryption Standard (AES), Publication 197.” http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf [retrieved: May, 2020].
5. ISO/IEC, “19772 Information technology - Security techniques - Authenticated encryption,” 2009.
6. P. Kocher, “Timing attacks on implementations of diffie-hellman RSA DSS and other systems,” in *Advances in Cryptology - CRYPTO ’96 Proceedings LNCS*, volume 1109, pp. 104-113 Springer Verlag, 1996.
7. P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Advances in Cryptology - Crypto 99 Proceedings LNCS*, volume 1666, pp. 388-397, Springer Verlag, 1999.
8. T. Krovetz and P. Rogaway, “The OCB authenticated-encryption algorithm, IETF RFC 7253,” May 2014.
9. K. Mayes and K. Markantonakis, editors. *Smart Cards, Tokens, Security and Applications*, chapter Chapter 17. Springer, 2nd edition, 2017.
10. K. Mayes, “Performance of Authenticated Encryption for Payment Cards with Crypto Co-processors,” in *Proc of ICONS17*, pp. 1-9, 2017.
11. MULTOS, http://www.multos.com/ [retrieved: May, 2020].
12. MULTOS, “The MULTOS Trust Anchor Development Board,” https://www.multos.com/dev_boards/devboard_details, [retrieved: May, 2020].
13. P. Rogaway, “OCB mode,” http://web.cs.ucdavis.edu/˜rogaway/ocb/ [retrieved: May, 2020].