Security Fusion Based On State Machine Compositions

Omar Al Ibrahim, Suku Nair
SMU HACNet Labs
Southern Methodist University
Dallas, TX, USA
{oalibrahim,nair}@smu.edu

***Abstract*—Security fusion is a new paradigm in security for resource-constrained environments [20]. Following this paradigm, strong system-level security is achieved by combining weak primitives from multiple nodes. In this paper, we describe a fusion methodology based on state machine compositions. From the properties of compositions, we devise a challenge-response system that composes low-entropy state machines at individual nodes into one with higher entropy. We use built-in digital logic such as Physical Unclonable Functions (PUFs) to efficiently mass generate and distribute keys. In addition, we draw on the properties of compositions to reduce the key storage complexity at the infrastructure-level, with high coverage and early detectability at the system-level.**

I. Introduction

Resource-constrained devices including sensors and Radio-Frequency ID (RFID) have posed severe limitations in processing power, memory and computational resources. These devices are subject to a multitude of attacks such as cloning and replay attacks. Traditional cryptographic schemes [1-3] are too expensive for these environments, naturally because of the stringent constraints on the device, but also because of the performance requirements on the system. In addition, resource-constrained systems usually hold a large set of nodes, and thus key management could be cumbersome.

A. Security fusion

Security fusion [20] is a novel framework in security for inherently resource-limited systems. Unlike traditional cryptographic schemes, the general approach here is to introduce lightweight primitives at the individual nodes that provide a strong aggregation to security properties at the system-level. In other words, security fusion refers to this concept of synthesizing security properties from weak point-to-point properties, much like the thin strands of a thick bulk rope.

This concept is suitable for protecting applications using a global security metric, without lending to strong cryptographic schemes. One application for security fusion is sensor networks [12-15]. Sensor networks consist of autonomous nodes that collect sensor measurements from the environment. These measurements are filtered and aggregated at some central higher-capable node to obtain improved information for system-level decisions. Due to the aggregate nature of data in sensor networks, it is natural to approach security through a fusion model to protect monitoring, tracking, and controlling applications. Using security fusion, it is possible to collate multiple sensor read-outs to reach a globally authentic decision.

Another application for security fusion is to preserve the component-level integrity of embedded systems. For instance, recent generations of smart phones consist of various interconnected components such as Bluetooth, WiFi, flash memory, and Codec chip. In many cases, these components are manufactured by different suppliers, and then integrated into a single platform. In the event of counterfeit, non-genuine components can be detected using an aggregate verification check of the platform.

Our earlier paper [20] introduces the concept of security fusion with a detailed description on a state machine-based architecture to realizing the framework. In this architecture, the basic idea is to have a challenge-response system that extracts a global security property from the response pattern of all the nodes. In [20], we analyzed the security characteristics of the proposed scheme and showed how it withstands a wide variety of attacks including cloning and intrusion.

B. State machine compositions

The purpose of this paper is to provide a fusion methodology for improved security and reduced complexity using state machine compositions [21,22]. For quite some time, researchers and engineers studied the mathematical properties of machine compositions and their applications. For instance in [23], machine compositions were used in fault-tolerant system design to create backups of the system state in a distributed client-server environment. Another application of the theory was presented in the simulation of digital logic circuits [24]. Machine compositions consider the problem of interconnecting an arbitrary set of state machines. These interconnections form logical associations that construct a state machine representation, called the *composite machine*. The interconnected state machines that constitute the composite machine are called the *component machines*.

Within state machine compositions, there are different types of interconnections

- *Cross product*: These are interconnections that execute machines concurrently.
- *Machine chains*: Machine chains are interconnections that execute in cascade fashion based on the propagation of

state information from one component to the next.
- *Feedback*: Interconnections by means of feedback composition propagate information flow from one component to a previous connected component.

Given these basic forms, we develop a fusion methodology based on the two former interconnection types: cross products and machine chains. Specifically, we use these interconnections to construct a high entropy model for system security using low-entropy state machines from individual nodes as building blocks. Moreover, we demonstrate a security-overhead tradeoff by transforming the composite machine into machine chains, which are then reduced to achieve the desired coverage.

The remainder of the paper is organized as follows. We briefly give an overview on security fusion in Section 2. Section 3 presents our fusion methodology using state machine compositions. In Section 4, we describe an approach to complexity reduction using state machine chaining. Finally, we conclude in Section 6.

II. Background - Security Fusion

In [20], we proposed a new paradigm to security, namely *security fusion*, in which security properties are synthesized from weak point-to-point primitives. The motivation is to present a common framework for security that is low-cost and secure at the system-level. In order to apply these concepts, it is necessary to establish a framework through which nodes will communicate with the backend system. In the next subsections, we present the physical architecture and a framework for security fusion using the model of a finite state machine.

A. Physical architecture

In the security fusion framework, the physical architecture consists of a system with large number of nodes (Figure 1). Each node is equipped with three components: computation, storage, and a communication component. The physical architecture also consists of a group of readers acting as cluster heads. The communication model is limited to a challenge-response interaction between the reader and the nodes. Thus, minimal communication is carried out and no delegation of messages is necessary. In the setup stage of the architecture, every node shares a secret key with the system. In this paper, we briefly discuss how the keys are efficiently derived and distributed using built-in circuitry such as Physical Unclonable Functions (PUFs) [30-35].

B. State machine model

We illustrate a simple example of security fusion using finite state machines. A finite state machine serves as a basic computational model and it is one of the many frameworks that could be utilized to apply security fusion concepts. Understanding the intrinsic properties of state machines helps us build systems with secure global properties.

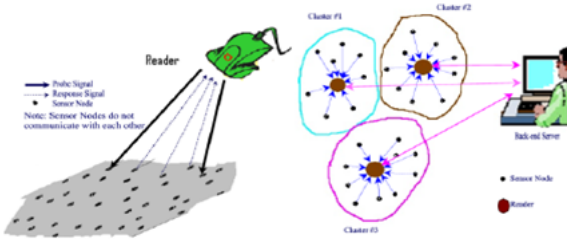


Fig. 1. Physical architecture

1) Description of the state machine: A state machine can be described by "transition" and "output" rules. Transition rules define the mappings from the current state to the next state given the input value. Output rules define possible values emitted by a node for a given state and transition. Let us assume we have a state machine with *n* states $(S_1, S_2, ...S_n)$. Since we are considering a state machine with a single-bit input, we have two transitions per state, one of which is "0" the other is "1". In the following, we describe the transition and output rules using a Moore model [22].

Transition rules: (Current state, Input) $\rightarrow$ Next state
- $(s_i, \text{"1"}) \rightarrow s_j$
- $(s_i, \text{"0"}) \rightarrow s_v$,

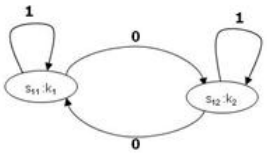
where $(0 \leq i, j, v \leq n)$
Output rules: (Current state) $\rightarrow$ Output
- $(s_i) \rightarrow k_i$
- $(s_j) \rightarrow k_j$, where $k_i \neq k_j$

To illustrate, we depicted three Moore machines M_1, M_2 and M_3 in the Figure 2 below. Each of these machines is associated with a node in the system.

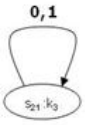
2) Security protocol: Each node is associated with a unique state machine and these state machines are shared as secret keys. The state machines are employed to model the read-out patterns. After each interrogation from the reader, nodes change their states according to the transition rules defined prior to deployment. The system authenticates a node by checking the expected current state and confirming correct application of the rules. A basic protocol for this interaction is depicted below.

Denote ***N***: node, ***R***: reader
- $R \rightarrow N$: Sends a read query
- N: Obtains the bit value (0 / 1)
- $N \rightarrow R$: N moves to the next state based on the bit value and sends a response
- R: resolves N's response

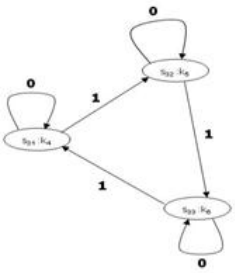
In the backend system, the values from multiple nodes are collated and processed by the application. During the execution of the state machine, a number from the current state is emitted. The numbers assigned to the states are random and unique. No synchronization is needed in the operation of the system, and we will simply assume that the start state is



a) Machine M_1



b) Machine M_2



c) Machine M_3

Fig. 2. Moore machine examples

provided as an argument in the query message. In this paper, we demonstrate how to use state machine compositions to increase the entropy of the system.

C. PUF-based implementation for state machines

In this architecture, we propose to use PUFs (Physical Unclonable Functions) [30-35] to efficiently derive and distribute state machines. PUFs are special circuits in semiconductor devices which provide unique challenge-response repository using device manufacturing variations. These responses have random but reproducible characteristics. PUFs are well known for their complex microstructure, and because of that they provide resistance to counterfeit. One of the distinct advantages of PUFs is that they are easily produced using standard digital logic. Thus, they can be used to mass manufacture random state machines at the individual nodes and then efficiently read-out by the system. PUFs are also efficient for implementation [4,6], and require much less hardware to implement than conventional cryptographic hash or encryption algorithms. For instance, existing designs of MD5 and SHA-256 require 8000 to 10000 gates [1,2,12]. In contrast, a PUF-based arbiter [32] circuit (Figure 3) is estimated to have 6 to 8 gates for each input bit, a total of about 545 gates for a 64-bit input PUF.

In this discussion, we briefly describe how to model a state machine using PUFs to leverage from this simple yet powerful circuitry. A general model for PUFs is to define a function that maps a set of challenges to a set of responses. PUF

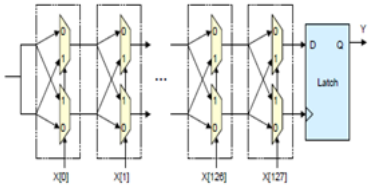


Fig. 3. Arbiter PUF circuit

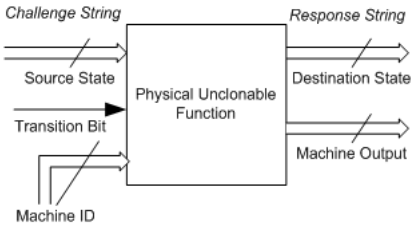


Fig. 4. State machine implementation using PUF

challenges and responses are represented as strings of bits. To model a state machine, we interpret the challenge-response strings as follows: In the challenge string, we define few bits to represent the source state, and another bit to denote the state machine transition. Since PUF has exponentially-many values with respect to the output length, we also specify a machine id to instantiate small state machines. In the response string, some of the bits will be used to determine the destination state, the rest are reserved for the machine output. In the execution of the state machine, PUF is used to dictate the transition and output rules. The system sends the current state and transition bit as part of the challenge string to PUF, and then obtains the next state and machine output from the response. Figure 4 captures the general idea.

Example: To illustrate, let us consider a state machine with four states using a 64-bit PUF circuit. First, we need to decide what lines will be used to represent the different fields. For instance, let us choose the first 61 bits of the input to denote the state machine id, and one bit to denote the transition, and the last two bits to decide on the current state. As for the response, we might also select the last two bits to represent the next state, the rest are for the state machine output. As a side note, we do not need to select the lines in consecutive order to represent the different fields since we can choose them at arbitrary positions. In Table I, we show a hypothetical example of challenge-response combinations using *91:D4:4B:29:20:80:56:70* as the machine id. The least significant hex digit in the challenge represents the state and transition bit. A challenge string *91:D4:4B:29:20:80:56:75* represents state s_1 on transition bit *1*. The corresponding response *3A:E6:CE:C1:CC:C9:C9:5B* represent output *3A:E6:CE:C1:CC:C9:C9:58* with state s_3 as the next state. A complete depiction of the state and transitioning behavior is shown in Figure 5.

TABLE I
CHALLENGE-RESPONSE MAPPINGS

Challenge	Response
91:D4:4B:29:20:80:56:70	92:00:11:EB:7F:E3:73:FF
91:D4:4B:29:20:80:56:71	B4:6D:17:DC:84:45:17:C1
91:D4:4B:29:20:80:56:72	0A:23:72:D1:DE:EB:ED:BD
91:D4:4B:29:20:80:56:73	B6:B2:DE:2A:10:00:56:20
91:D4:4B:29:20:80:56:74	F4:F7:F7:F7:76:83:12:20
91:D4:4B:29:20:80:56:75	3A:E6:CE:C1:CC:C9:C9:5B
91:D4:4B:29:20:80:56:76	84:45:17:C1:65:FD:85:81
91:D4:4B:29:20:80:56:77	3A:E6:CE:C1:19:F3:20:E6

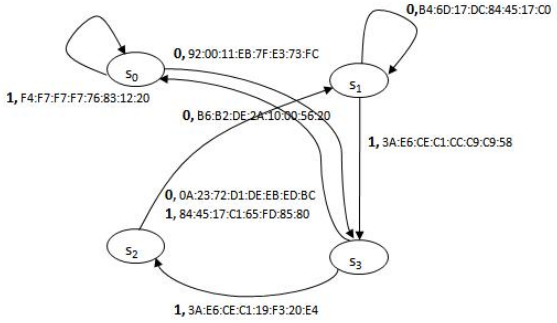


Fig. 5. State machine example for PUF mappings in Table I

III. STATE MACHINE COMPOSITIONS

In the following sections, we present the security fusion architecture based on compositions, but first let us introduce the notations/terminology used in this paper (Table II).

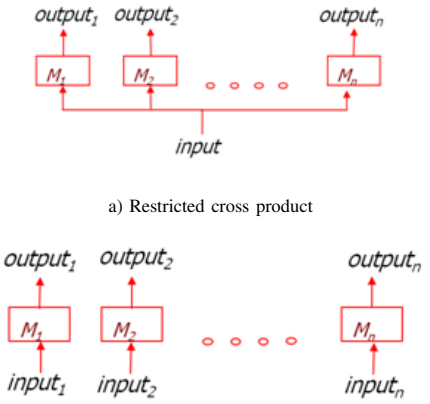
TABLE II
NOTATIONS

Symbol	Description
M_i	Moore state machine, ferived from the nodes.
P	Product machine, constructed using a composition of multiple Moore machines.
C_i	Machine chain component.
s_{ij}	State j in some Moore machine M_i.
x_{ij}	State j in some machine chain component C_i.
p_j	State j in the product machine P
$(s_{1j_1}, s_{2j_2}, ...,)$	Composition state representation of the product machine state p_j . Each element of the tuple is a state in a Moore machine. The tuple length depends on the composition size.
$k_1, k_2, k_3, ..$	Node responses which are governed by the output rules on the Moore machines. These responses are generated and assigned uniquely to the Moore machines.
$s_{ij} : k$	An assignment of a node response k to state s_{ij}

A. Cross product

A cross product machine [21,22] is a state machine that simulates concurrent execution of multiple machines. In other words, each state in the cross product machine represents a configuration of states for a group of machines. Two cross product models simulate parallel execution of state machines:

- *Restricted cross product*: The restricted cross product is a machine construction that simulates execution of multiple machines fed with the same input.
- *Full cross product*: The full cross product is a machine construction that simulates execution of multiple machines for all input combinations. This is the more general construction.



a) Restricted cross product

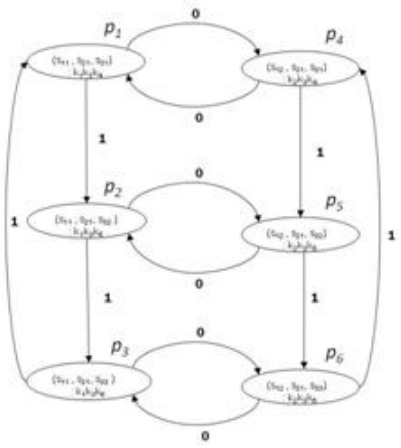
b) Full cross product

Fig. 6. Cross product models

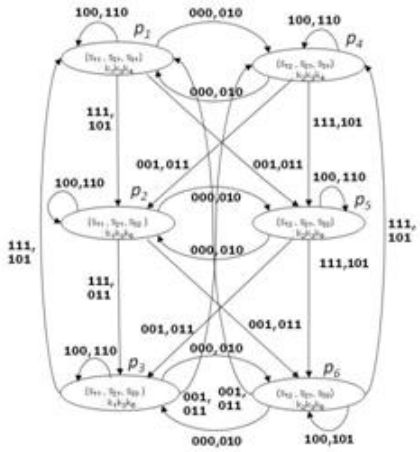
A logical diagram for both models is depicted in Figure 6. Both constructions consider the reachable product of states for some arbitrary number of machines $(M_1, M_2 \ldots M_k)$. A state p_j is said to be reachable if and only if there exists a sequence of transitions that takes the machine to p_j starting from the initial state.

Example: Consider the three Moore machines M_1, M_2 and M_3 shown in Figures 2(a), (b), and (c). The restricted and full cross products for the three machines are depicted in Figures 5(a) and (b) and denoted as P. To construct P, we first determine the set of all reachable states in each machine. In this example, the sets are: $\{s_{11}, s_{12}\}$ for $M_1, \{s_{21}\}$ for $M_2, \{s_{31}, s_{32}, s_{33}\}$ for M_3 . Second, we generate the product states by taking all different combinations from the sets: $\{(s_{11}, s_{21}, s_{31}), (s_{11}, s_{21}, s_{32}), (s_{11}, s_{21}, s_{33}), (s_{12}, s_{21}, s_{31}), (s_{12}, s_{21}, s_{32}), (s_{12}, s_{21}, s_{33})\}$.

As illustrated in Figure 7, we represent the product states p_1 through p_6 as a tuple of the original states. For instance, state p_1 of the cross product has a composite representation (s_{11}, s_{21}, s_{31}) which represents a configuration of M_1 in s_{11}, M_2 in s_{21}, and M_3 in s_{31}. To compute the transitions for the cross product, we evaluate the transition rules of each state in the tuple. In the restricted cross product, the transition rule is evaluated for a common input. For instance, if the input is



a) Restricted product *P*



b) Full product *P*

Fig. 7. Cross product examples

0, then M_1 moves to s_{11} and outputs k_1, M_2 moves to s_{21} and outputs k_3, and M_3 moves to s_{31} and outputs k_4. Therefore, the next state in the product machine will be (s_{12}, s_{21}, s_{31}), and corresponds to p_4, and the response pattern becomes $k_2k_3k_4$(as a simple convention, we represent the pattern as a string). As for the full cross product, the transition rules are evaluated by taking all input combinations.

B. Authentication using the cross product

We now describe an authentication procedure using the cross product. The cross product is used to compute a system-wide output (e.g. using XOR) from the response pattern of elemental state machines stored at the individual nodes. We illustrate this with a simple example of authenticating three nodes using $M_1, M_2,$ and M_3 in Figure 2.

Example: Consider a scenario in which $M_1$is in s_{11}, M_2 is in s_{21}, and M_3 is in s_{31} (Figure 8). In this example, assume the reader input is *1*, and also assume that we are going to apply the restricted cross product, as illustrated in Figure 7(a). The corresponding responses for this configuration are k_1 for

TABLE III
CROSS PRODUCT MAPPINGS

State in product machine P	Composite representation	Response pattern
p_1	(s_{11}, s_{21}, s_{31})	$k_1k_3k_4$
p_2	(s_{11}, s_{21}, s_{32})	$k_1k_3k_5$
p_3	(s_{11}, s_{21}, s_{33})	$k_1k_3k_6$
p_4	(s_{12}, s_{21}, s_{31})	$k_2k_3k_4$
p_5	(s_{11}, s_{21}, s_{31})	$k_2k_3k_5$
p_6	(s_{11}, s_{21}, s_{31})	$k_2k_3k_6$

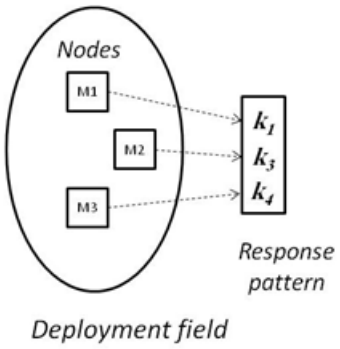


Fig. 8. Response pattern example

M_1, k_3 for M_2, k_4 for M_3. Collectively, the reader obtains $k_1k_3k_4$ as a response pattern (Figure 8). Table 1 illustrates the mappings for every response pattern to a product state in the system. As shown from the table, the pattern $k_1k_3k_4$ corresponds to state p_1 in the product machine, and the initial states (s_{11}, s_{21}, s_{31}) correspond to p_6. Accordingly, the system accepts the response pattern since $(p_6, 1) \rightarrow p_1$ is a valid transition in the product machine. On the contrary, consider a scenario in which M_1 is replaced with a malicious node that outputs k_2 instead of k_1. In this case, the system obtains $k_2k_3k_4$ as a response pattern instead of $k_1k_3k_4$. Referring to the table, we deduce that the new response pattern corresponds to p_4 and since the transition $(p_4, 1) \rightarrow p_1$ is not a valid transition, the response pattern is rejected by the system.

IV. COMPLEXITY REDUCTION USING MACHINE CHAINING

In this section, we describe a transformation called machine chaining to reduce the state space incurred from the cross product construction. State machine chaining is a composition that interconnects a set of state machines as components in a chain. The chain components execute in cascade fashion to simulate a larger composite machine. In this section, we explain the details of the transformation and how to establish an authentication procedure based on this new representation.

A. Machine chaining

Machine chaining [21,22] is a construction of multiple state machines logically connected in cascade form. In this interconnection, every state machine is treated as a component such that the state of one component depends on the state of the previous component in the chain. The first component, C_1, depends only on the input sequence, and thereby called the *independent component*. The rest of the components, $C_2 \ldots C_m$, determine their state based on the influence of the previous components, and thereby referred to as *dependent components*.

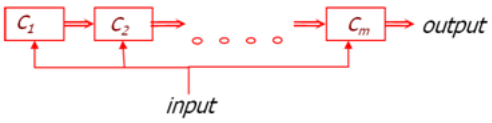


Fig. 9. Machine chaining

The theory of machine chaining has been around for decades [21,22] yet the scope of applications are limited to the design of sequential circuits. For instance, using machine chaining, a large circuit is replaced with an interconnection of small sub-circuits. These small sub-circuits are synthesized to reduce costs and to provide reliability advantages including ease of trouble-shoot and repair. In this paper, we demonstrate a new application for machine chaining in the areas of security and system design. Theoretically, it has been demonstrated that any state machine has a chain realization [22] such that the components of this chain are "algebraically" simpler than the original state machine.

Example: We illustrate a machine chain with two components: an independent component (C_1) and a dependent component (C_2). This chain has equivalent behavior as the product machine P shown in Figure 7(a). Depicted above, C_1 has two states x_{11} and x_{12}. Since C_1 is independent, the transitions are only triggered by the machine input. C_2, on the other hand, depends on both the input and the state of the previous machine. As illustrated, the state delegation from C_1 to C_2 is treated as part of the input (Figure 10(b)). Referring to Figure 10(b), consider C_2 being on state x_{21}. If the input is *1* and C_1 is on x_{11}, then C_2 will transition to x_{22} and outputs $k_1k_3k_5$. Instead if the input was *0*, then it would have output $k_1k_3k_4$. This output behavior is equivalent to p_1 in the product machine P. Similarly, we find matching equivalences for the rest of the states. As a general rule, if two machines have the same behavior, then there must be a correspondence between their states. Therefore, every state in P has a matching configuration of the component machines, represented as a composite vector of the component states. Table IV shows this correspondence.

B. Chain transformation

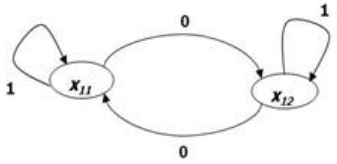
We now describe the setup procedure for building a security fusion architecture using state machine chaining. The system pre-computes a machine chain to transform elemental state machines into a single representation. Using the properties of machine chains, the system machine is expressed as a vector of interconnected components. Subsequently, the coverage of the system is determined by the number of components that represent the system state. To obtain a coverage-overhead tradeoff, we propose to reduce the chain length by eliminating some of the components from the chain. The coverage of the system is chosen to reduce the chain length while keeping the likelihood of intrusion as low as possible.

Let us investigate how to transform the individual state machines into a chain construction (Figure 11). This transformation is computed in the setup phase of the system. We can describe the setup procedure in the following steps:

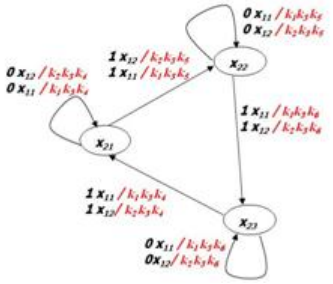
- Step 1: Initially, the system reads-out the state machines for all the nodes. The state machines $(M_1, M_2 \ldots M_n)$ are derived from the nodes and stored in the system.
- Step 2: Next, we construct a cross product P from the machines $(M_1, M_2 \ldots M_n)$. Although the size of the cross product could be potentially large, the cross product is only stored temporarily.
- Step 3: P is factored out into a machine chain $(C_1, C_2, \ldots, C_m)$. Various chaining algorithms perform

TABLE IV
STATE CORRESPONDENCE MAPPINGS

State in product machine P	State in C_1	State in C_2
p_1	x_{11}	x_{21}
p_2	x_{11}	x_{22}
p_3	x_{11}	x_{23}
p_4	x_{12}	x_{21}
p_5	x_{12}	x_{22}
p_6	x_{12}	x_{23}



a) Independent component C_1



b) Dependent component C_2

Fig. 10. Machine chain components

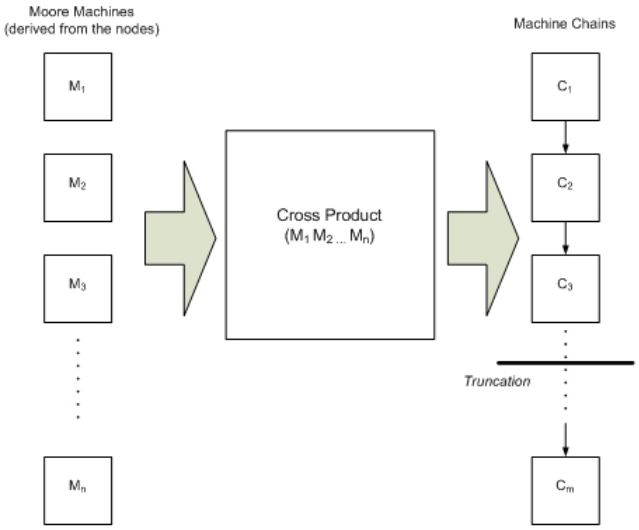


Fig. 11. Chain transformation and truncation

these conversions, as we will describe shortly.

- Step 4: We truncate the chain at some cutoff point. The choice for the truncation point is chosen to achieve an optimal coverage-overhead tradeoff.

C. Authentication through chaining

Example: To illustrate the concept, we continue on our example from Figure 8 and the three state machines M_1, M_2 and M_3 shown previously in Figures 2(a), (b) and (c), but rather than authenticating with P , we transform P into a machine chain with two components C_1 and C_2, as illustrated in Figures 10(a) and (b). According to Tables 1 and 2, the corresponding states for $k_1k_3k_4$ will be x_{11} in C_1, and x_{21}in C_2. Further, suppose the initial system state is x_{11} in C_1 and x_{23} in C_2. Since the outputs satisfy the transition rules $(x_{11}, 1) \rightarrow x_{11}$ and $(x_{23}, 1) \rightarrow x_{21}$, the response pattern is accepted. On the other hand, consider a scenario in which a malicious node masquerades as M_1 and suppose it emits k_2 instead of k_1 . In this scenario, the response pattern $k_2k_3k_4$ will correspond to states $x_{12}x_{21}$, mapping to an incorrect transition $(x_{12}, 1) \rightarrow x_{11}$in C_1. As a result, the system detects a false response pattern.

D. Chaining approaches

Several chaining approaches were proposed to transform a state machine into a machine chain. This transformation produces a state machine of equivalent behavior. Here we briefly describe a few of these approaches and the structure of the components involved in the factorization. To make discussions easier, let us consider factoring P into two components C_1 and C_2. The general case is straightforward to deduce.

1) Delayed input chaining: One approach to chaining is to factor out a state machine by another machine that follows the behavior of a delay line circuit [22]. In electronics, a delay line is defined as a series of connecting delay elements, i.e. latches, in which the output line matches the input line after multiple time intervals. The state machine corresponding to this circuit is used to synthesize C_2.

2) Chaining with arbitrary components: Delayed-input chaining [22] does not always lead to complete factorization. An alternative approach is to select an arbitrary structure for C_1. In this approach, a restricted cross product of C_1 and P is constructed to determine a state cover. C_2 is synthesized by factoring out the component states from the cross product using the state cover.

3) Chaining with PR-machines: A more systematic approach to factorization is to construct a chain of Permutation-Reset (PR) machines [22]. PR-machines are state machines that satisfy the following property: All transitions from a state either permutes to all states or resets to one particular state. Figure 10(a) is an example of a PR-machine showing a permutation on states: $x_{11}$and x_{12}. PR-machines are useful structures because they are guaranteed to factor out any state machine. Formally, it can be shown that any *n*-state machine is factorable into an independent component that is a PR-machine followed by a dependent component having *n-1* or fewer states.

V. Conclusion

In this paper, we developed a security fusion methodology based on state machine compositions. We demonstrated that compositions such as cross products and machine chains provide improved authentication with reduced complexity for an infrastructure. We also discussed how built-in primitives such as PUF could be used to improve distribution of keys among the nodes. Future work involves analytical study of overhead to security tradeoff available to the proposed methodology.

References

[1] A Bogdanov, G Leander, C Paar, A Poschmann, and Y Seurin, *Hash functions and RFID tags: mind the gap,* in CHES 2008 - Cryptographic Hardware and Embedded Systems , 2008, pp. 283-299.

[2] M. Feldhofer and C. Rechberger, *A case against currently used hash functions in RFID protocols,* in OTM 2006 Workshops, 2006, pp. 372-381.

[3] D. Wheeler and R. Needham, *TEA, a tiny encryption algorithm,* in Fast Software Encryption: Second International Workshop, Lecture Notes in Computer Science , 1994, p. 363366.

[4] H. Chan, V. Gilgor, A. Perrig, and G. Muralidharan, *On the distribution and revocation of cryptographic keys in sensor networks,* IEEE Transactions on Dependable and Secure Computing , pp. 233-247 , July 2005.

[5] W. Du, J. Deng, Y. Han, and P. Varshney, *A key predistribution scheme for sensor networks using deployment knowledge,* IEEE Transactions on Dependable and Secure Computing , pp. 62-77, January 2006.

[6] N. Saxena, G. Tsudik, and J. Yi, *Efficient node admission and certificate-less secure communication in short-lived MANETs,* IEEE Transactions on Parallel and Distributed Systems , pp. 158-170, February 2009.

[7] E. Shai and A. Perrig, *Designing Secure Sensor Networks,* IEEE Wireless Communication, vol. 11, no. 6, pp. 38-43, December 2004.

[8] A. Uluagac, R. Beyah, Y. Li, and J. Copeland, *VEBEK: Virtual Energy-Based Encryption and Keying for Wireless Sensor Networks,* IEEE Transactions on Mobile Computing , pp. 994-1007 , July 2010.

[9] Z. Yu and Y. Guan, *A Key Management Scheme Using deployment knowledge for wireless sensor networks,* IEEE Transactions on Parallel and Distributed Systems , pp. 1411-1425 , October 2008.

[10] C. Karlof, N. Sastry, and D. Wagner, *TinySec: A link layer security architecture for sensor networks,* in Proceedings of the Second ACM Conference on Embedded Networked Senosr Systems (SenSys), 2004.

[11] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J.D. Tygar, *SPINS: Security protocols for sensor networks,* in Proceedings of Seventh Annual International Conference on Mobile Computing and Networkings, 2001.

[12] M. Feldhofer and J. Wolkerstorfer, *Strong crypto for RFID tags - a comparison of low per hardware implementations,* in IEEE International Symposium on Circuits and Systems, 2007, pp. 1839-1842.

[13] R.I. Praise and S. Vaudenay, *Mutual authentication in RFID: security and privacy,* in ACM Symposium on Information, Computer and Communications Security, New York, NY, USA, 2008, pp. 292-299.

[14] L. Yan, Y. Zhang, L. Yang, and H. Ning, *The internet of things*. Boca Raton, FL: Auerbach, 2008.

[15] M Burmester, T van Le, and B de Medeiros, *Provably secure ubiquitous systems: universally composable RFID authentication protocols,* in Securecomm and Workshops, 2006, Baltimore, MD,USA, 2006, pp. 1-9.

[16] P. Israsena, *Securing ubiquitous and low-cost RFID using Tiny Encryption Algorithm,* in 1st International Symposium on Wireless Pervasive Computing, 2006.

[17] Y. Yu, Y. Yang, and H. Min, *A Novel Design of Secure RFID Tag Baseband,* in Proceedings of EU RFID Forum, 2007.

[18] K. Youngdai, S. Hong, W. Lee, S. Lee, and J. Lim, *Related key differential attacks on 26 rounds of XTEA and full rounds of GOST,* in Proceedings of Fast Software Encryption, Lecture Notes in Computer Science, 2004, pp. 299-316.

[19] L. Blum, M. Blum, and M. Shub, *A Simple unpredictable pseudo-random number generator,* SIAM Journal on Computing, vol. 15, p. 364383, May 1986.

[20] S. Nair, S. Abraham, and O. Al Ibrahim, *Security fusion: security for resource-constrained environments,* in IEEE Infocom, Shanghai, China, 2011 (submitted).

[21] W. Holcombe, *Algebraic automata theory*, 2nd ed. New York, USA: Cambridge University Press, 1982.

[22] F. Hennie, *Finite-state models for logical machines*, 4th ed. New York, USA: John Wiley Sons, 1968.

[23] B. Balasubramanian, V. Ogale, and V. Garg, *Fault tolerance in finite state machines using fusion,* in Distributed Computing and Networking, Lecture Notes in Computer Science.: Springer, 2008, pp. 124-134.

[24] P. Maurer, *Logic simulation using networks of state machines,* in Proceedings of the conference on Design, automation and test in Europe , Paris, France, 2000, pp. 674 - 678.

[25] J. Farnendez and A. Farnedez, *SCADA systems: vulnerabilities and remediation,* Journal of Computing Sciences in Colleges, vol. 20, no. 4, pp. 160-168, April 2005.

[26] *Critical infrastructure protection: challenges and efforts to secure control systems,* General Accounting Office (GAO), GAO-04-354, 2004.

[27] N. Ferguson and B. Schneier, *Practical cryptography*: Wiley publishing, 2003.

[28] T. Cormen, C. Lieserson, R. Rivest, and C. Stein, *Introduction to algorithms*, 2nd ed.: MIT Press and McGraw-Hill, 2001.

[29] A. Maiti, R. Nagesh, A. Reddy, and P. Schaumont, *Physical unclonable function and true random number generator: a compact and scalable implementation,* in Proceedings of the 19th ACM Great Lakes symposium on VLSI , Boston Area, MA, 2009, pp. 425-428.

[30] M. Majzoobi, F. Koushanfar, and M. Potkonjak, *Techniques for design and implementation of secure reconfigurable pufs,* ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 2, no. 1, March 2009.

[31] A. Maiti, R. Nagesh, A. Reddy, and P. Schaumont, *Physical unclonable function and true random number generator: a compact and scalable implementation,* in Proceedings of the 19th ACM Great Lakes symposium on VLSI , Boston Area, MA, 2009, pp. 425-428.

[32] G. Suh and S. Devadas, *Physical unclonable functions for device authentication and secret key generation,* in ACM IEEE Design Automation Conference, San Diego, CA, 2007, pp. 9-14.

[33] L. Bolotnyy and G. Robins, *Physically unclonable function-based security and privacy in Rfid systems,* in Fifth Annual IEEE International Conference on Pervasive Computing and Communications, 2007. PerCom '07. , White Plains, NY, 2007, pp. 211-220.

[34] L. Kulseng, Z. Yu, Y. Wei, and Y. Guan, *Lightweight secure search protocols for low-cost Rfid systems,* in 29th IEEE International Conference on Distributed Computing Systems, 2009, pp. 40-48.

[35] K. Frikken, M. Blanton, and M. Atallah, *Robust authentication using physically unclonable functions,* in Lecture Notes in Computer Science.: Springer Berlin / Heidelberg, 2009, pp. 262-277.