State Machine-Based Security Fusion for Resource-Constrained Environments

S.Nair, O.Al Ibrahim, and S.Abraham
SMU HACNet Labs, Southern Methodist University, Dallas, TX

***Abstract*—A growing range of devices have difficulty in implementing strong cryptographic algorithms. RFIDs and sensors, for instance, generally lack the processing power and memory to perform these operations in an efficient and timely manner. Recently, a new paradigm in security, namely *security fusion* [24], was introduced for resource-constrained environments. In this approach, strong security properties are synthesized from weaker point-to-point properties, thereby minimizing the resource requirements at each node without compromising the system-level security. In this paper, we describe a state machine-based architecture and pertinent protocols to realize security fusion. Further, we analyze these protocols for their security capabilities.**

***Index Terms*—security, RFID, sensor, state machines.**

I. Introduction

Recently, wireless sensors and RFID (Radio Frequency ID) have gained a lot of popularity due to their wide range of applications in surveillance and tracking. Though these devices have provided promising solutions for various applications, they are subject to a multitude of attacks including eavesdropping, intrusion and replay attacks.

Implementing security in sensors and RFID have posed great challenges given the severe limitations in processing power, memory and computational cycles. For instance, a passive RFID tag that complies with the EPC C1G2 standard [1] has a baseband of 7500 Gate Equivalence (GE) out of which 200-2000 gates are reserved for security functionalities [27] [29]. The chip area for such tags is approximately 0.8mm^2 and the power consumption is roughly 30 μW with EEPROM programming [16]. Wireless sensors are also limited in processing speed, storage capacity and power. Popular sensor devices have RAM sizes ranging from 4 KB on the Mica2 platform to 10KB on the TelosB platform [10]. The storage capacity is usually between 4KB and 512KB and may come incorporated with Bluetooth, Chipcorn CC100, or IEEE 802.15.4 compliant transceivers. Today, devices such as micro-sensors and nano-sensors are increasing in popularity over conventional sensors. These sensors are known for their extremely small size, low cost, and high sensitivity.

Therefore, supporting cryptography has become challenging given these characteristics. Table I ([28]) provides a comparison between various cryptographic algorithms with respect to the number of cycles, gates, and average power consumption.

This work is supported in part by a grant from NIST and in part by a grant from Globeranger as a project in conjunction with the NSF NetCentric Software & Systems Industry/University Cooperative Research Center.

TABLE I
Cost Analysis of Various Algorithms

Algorithm	Key Bits	Plaintext	Cycles	GE	Power (μW)
AES	128	128	1016	3595	8.15
TEA	128	64	64	2355	12.34
SHA-1	*L*	192	405	4276	26.73
LFSR	32	64	92	685	0.1582
DES	56	64	144	2309	2.14
ECC	113	*L*	195159	10k	*L*

L denotes number of bits

Generally speaking, the complexity involved with traditional cryptography is formidable for low-cost sensors and RFID. Despite Moore's Law, sensors and RFID will remain resource-constrained because of their design. On top of that, new advancements in process technologies will be insufficient for tighter constraints in the new generations of these devices.

A. Security Fusion

In [23], a new architecture for incorporating security in sensors and RFID environments was introduced. The goal of this architecture is to provide strong system-level security from the application of many weak security primitives. The term *security fusion*, was introduced in [24] to refer to the broad motivation of this architecture.

The principle of security fusion is that nodes do not provide all the protection by themselves, since they only implement lightweight primitives, but unlike traditional schemes, these primitives are fusible to provide greater security, much like the thin strands of a thick bulk rope. The earlier paper [24] proposed an instantiation of this idea based on a simple secret state machine on each device, which is shared by a reader. In [4], an extension of the work presented a fusion methodology based on state machine compositions [14]. From the properties of compositions, the authors define methods of combining state machines to provide global authentication and reduced complexity for environments that cannot support a complete security schema.

This paper is intended to present a state-machine based security fusion architecture. We compile previous results and expand on the security analysis and experimental evaluation.

B. Applications

Security fusion is a new paradigm in security, where the aggregate system is strong even though individual elements use lightweight security. The proposed application spaces of security fusion are certainly of growing issues to the security community. Here, we mention a few of them.

- *Low-cost RFID*: Security has become a rising concern for RFID mainly because scanning and replicating tags require little money or expertise. While cryptographic RFID purports to offer seemingly strong protection, they are often too expensive to be used in large quantities. To overcome factors such as power budget, computational latency, and cost, security fusion enables simple mechanisms to be incorporated in RFID tags, which can be consolidated quickly by the middleware. With multi-tag RFID systems [7], security fusion becomes more effective.
- *Sensor networks*: Sensor networks consist of autonomous nodes that collectively obtain measurements from the environment to improve system-level decisions. Due to the aggregate nature of data in sensor networks, it is natural to define security through a fusion model that protects the monitoring, tracking, and controlling functions of an application. Security fusion has a lot of potential in sensor networks. Using security fusion, it is possible to collate multiple sensor read-outs to reach a globally authentic decision.
- *Embedded systems*: Another potential 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 radio, WiFi radio, flash memory, and Codec chip. These components are manufactured by different suppliers and then integrated into a single platform. In the event of counterfeiting, non-genuine components can be detected through security fusion using an aggregate verification check of the platform.

The remainder of the paper is organized as follows. In Section 2, we discuss previous research. In Section 3, we explore the security fusion paradigm in more detail. In Sections 4 and 5, we describe a state machine-based architecture for security fusion. Analysis of the security characteristics is presented in Section 6. Then in Section 7, an experimental evaluation is demonstrated. Finally, we conclude the paper in Section 8.

II. Previous Work

A couple of security architectures have been proposed to protect sensor networks from various attacks. Perrig et al. [20] introduced an architecture called SPINS which consists of two building blocks: SNEP - a security protocol that employs symmetric cryptography using RC5 for encryption and Message Authentication Code (MAC) for integrity, as well as μTESLA - which provides authentication for data broadcast. The SNEP protocol is designed for resource-constrained sensor networks. To encrypt messages in the SNEP protocol, each sensor node shares a master key with a base station, from which it derives one-time encryption keys to be fed to RC5. The other building block of SPIN, μTESLA, provides authenticated broadcast for sensor networks. μTESLA was designed based on the standard TESLA protocol, but is developed to address limited computing environments.

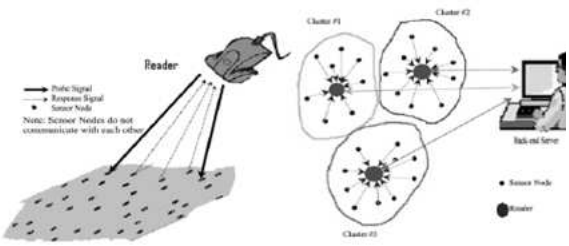


Fig. 1. Physical architecture. Left: a reader sends a probe signal to nodes in a deployment field; nodes respond in reverse unicast. Right: Architecture organized in a cluster network with readers acting as cluster heads.

Another security architecture that has recieved a swirl of attention in the past few years is TinySec. TinySec, proposed by Wagner et al. [17], is a link layer security architecture for micro-sensor networks which is tightly coupled with Berkeley TinyOS. The motivation for TinySec was to increase the network resistance against denial-of-service attacks by enabling early detection of unauthorized packets in the network, and preventing them when they are first injected as opposed to waiting for packets to route to the destination, thus saving energy and bandwidth.

III. Security Fusion

The general approach of security fusion is to introduce lightweight elements at individual nodes which provide a strong aggregation of security. In this section, we describe some of the theoretical foundations of these concepts. We implicitly discuss the concept of security fusion in-line with the problem of achieving authentication in a decision system. However, the broad motivation extends to other security attributes including integrity and confidentiality.

It is important to note that security fusion is not just a framework for secure consensus among nodes, but rather it extends to fusion of security as a property. Unlike trusting crowded-sourced nodes, security fusion is based on a centralized challenge-response system which aggregates the security characteristics of multiple nodes. In this paper, we present an instantiation of the security fusion concept based on the model of a finite state machine. But first, we start the discussions by describing the physical architecture and the attack model.

A. Physical Architecture

The physical architecture consists of a typical centralized cluster-based network comprised of a large number of nodes (Figure 1). Each node is equipped with three components: computation, storage, and a communication. 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 performed in these nodes and no

delegation of messages is necessary. In the setup stage of the architecture, every node shares a secret key with the system. Further, the keys are assumed to be distributed to the nodes before deployment.

B. Attack Model

In the security fusion architecture, we consider an attacker that is adaptive across space and time. Given these characteristics, we identified a number of attacks that can be launched:

- *Malicious read*: With a rogue reader in possession, an attacker may attempt to read out from a node. By interrogating repeatedly, he may collect packets to derive the secret.
- *Brute-force attacks*: An attacker may expand his attack by exhausting all possible responses for a given node. He may listen passively to a communication or actively read out to collect packets.
- *Replay attacks*: An attacker may collect packets and replay them back to the system at a later time. The success of the attack will depend on whether an attacker can replay the correct responses to fool the system.
- *Physical capture*: A determined attacker may go as far as capturing a node to extract the shared secret. We assume that compromised nodes will not reveal the secrets of other nodes as no two nodes will share the same secret.
- *Replication*: In an attempt to spoof the responses, an attacker may replicate a node and insert it into the network hoping to disrupt the system.

C. State Machine-Based Security Fusion

In order to apply the concept of security fusion, it is necessary to establish a framework using which the nodes will communicate with the backend system. One of the requirements of this framework is that it has to be very lightweight, general and useful across multiple application areas. In this paper, we present a security fusion architecture based on the model of a finite state machine. Finite state machines serve as a basic computational model that can help build systems with secure global properties.

In this architecture, each node will have a unique state machine that is shared as a secret key with the backend system. The underlying state machine governs the output pattern of the node by following certain transition rules. Furthermore, each state is assigned a disjoint set of uniquely and randomly-generated values, or *pseudonyms*, which represent the outputs. The input to the state machine is internally derived and can be used to express something like sensor detection results (as boolean logic detection/ non-detection in the case of a sensor node).

Based on this, we present two fusion methods to attain security. First, we describe a simple fusion algorithm using majority logic. Next, we advance to stronger fusion methods based on the properties of state machine compositions.

1) Majority logic: In the first fusion method, the nodes communicate a boolean logic to the reader by randomly selecting a pseudonym from a set of possible pseudonyms assigned to the current state before the state machine transitions to the next state. Then, the outputs are forwarded to the backend system and decoded to reach a consensus. The security properties derived from this consensus are hard, i.e. they are non-trivially amplified with increased number of nodes.

In the analysis, we demonstrate that the state machine stored in each node is deterministic. However by using the pseudonyms, the obfuscation converts the deterministic state machine (DFA) into one which is larger in size and non-deterministic (NFA) from an attacker's perspective. For a certain selection of the state machines, we can show that the NFAs are cumbersome to reduce by minimization algorithms, especially when the problem size is scaled to a large number of nodes. These non-deterministic properties are used to build a strong fusion property for security.

2) State machine compositions: State machine compositions [14] [15] consider the problem of interconnecting an arbitrary set of state machines to contruct a single machine. For quite some time, researchers and engineers studied the mathematical properties of compositions and their applications. For instance, state machine compositions were used in fault-tolerant system design to create backups of the system state in distributed client-server environments [5]. Another application of the theory was presented in the simulation of digital logic circuits [19].

In this approach, we use state machine compositions to keep track of a system-wide output by combining a set of low-entropic machines into one with higher entropy. These compositions come in several different types. Based on these composition types, we develop a fusion methodology to improve system-level security and reduce overall complexity.

IV. Security Fusion Based On Majority

In the following section, we describe the fusion approach based on majority logic of state machines. For the purpose of illustration, we consider the model of a finite state machine with two outgoing transitions per state, triggered on binary input (0/1).

A. 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: "0" and "1".

In the following, we describe the transition and output rules using a Mealy model [14].
Transition rules: (Current state, Input) $\rightarrow$ Next state

- $(s_i, 0) \rightarrow s_j$
- $(s_i, 1) \rightarrow s_v$, where $(0 \leq i, j, v \leq n)$

Output rules: (Current state, Input) $\rightarrow$ Output

- $(s_i, 0) \rightarrow k_j$
- $(s_i, 1) \rightarrow k_v$, where $k_j \neq k_v$

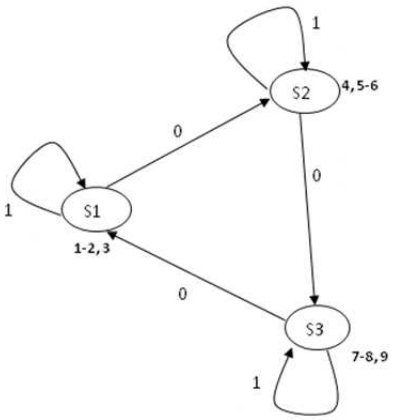


Fig. 2. State machine diagram. Pseudonyms 1 through 9 are distributed among states s_1, s_2, and s_3. In each state, the pseudonyms are split between the two transitions.

Consider assigning *k* unique pseudonyms to each state in the state machine, of which $p(1 \leq p \leq k)$ pseudonyms are used to represent a transition on (0) and $q(q = k - p)$ pseudonyms are used to represent a transition on (1). In the execution of the state machine, a node randomly selects a pseudonym from either set *p* or *q* assigned to the current state. Then it transmits the pseudonym and transitions to the next state using the transition rules. As an example, consider a 3-state finite state machine

1) *n*=3 $\{s_1, s_2, s_3\}$
2) *k*=3 [Each state is assigned a set of 3 numbers of which $p(1 <= p < k)$ numbers may be used to represent (0) and $q = k - p$ numbers may be used to represent a (1).]
3) The total set of numbers available for the 3-finite state machine = *nk* = 9
4) Each state $\{s_1, s_2, s_3\}$ will have *k*=3 numbers assigned to it.

Figure 2 illustrates the state diagram and Table II shows an example of a pseudonym assignment for the state diagram. In the next section, we describe the protocol and authentication procedure using the state machine model.

B. Security Protocol

In the security protocol, each node is associated with a unique state machine which is shared as a secret key with the backend system. After each interrogation from the reader, nodes may change their states according to the transition rules defined prior to deployment. The reader verifies the legitimacy of a node by checking the expected current state and confirming the correct application of the rules.
A basic protocol for this interaction is depicted below.
Denote *N*: node, *R*: reader

1) *R* $\rightarrow$*N: Send read query*
2) *N: Obtain the bit value (0/1)*
3) *N* $\rightarrow$ *R: N moves to the next state based on the bit value and outputs a pseudonym from set p or q*
4) *R resolves N's output and re-syncs*

It should be noted that even though the values read from each node are similar to a point-to-point security channel, no security properties are derived from just one value. Rather, a large number of the read values are collated to derive the security fusion properties.

In order to properly authenticate a node, the backend system needs to maintain a copy of the state machines associated with all the nodes in the network. These state machines are indexed by the corresponding node id. When interrogated by a reader, a node responds with a pseudonym and possibly with its id. This value of the pseudonym depends on the state machine rules defined prior to deployment. Since the system can obtain the node's state and the pseudonyms assigned for that state, it could simply authenticate a node by the pseudonym. The backend system can also deduce the original value of the node using the mapping between the pseudonyms and the actual information. After getting the response, the system will also update its copy of the state machine corresponding to that node with the new state value.

On the back-end server, the backend system assigns a unique node id and stores flag bits to track the current state of the node. For every state and machine input, the server also stores the next state and pseudonym set. Whenever the server receives a response, it checks if the value matches any of the elements in the pseudonym set of the node's current execution. Otherwise, the response is rejected.

TABLE II
PSEUDONYM ASSIGNMENT EXAMPLE

States	p Transition on 0	q Transition on 1
s_1	1 or 2	3
s_2	4	5 or 6
s_3	7 or 8	9

TABLE III
MACHINE INDEX TABLE

Node id	Flag	Current state	Next state/Output	
			i=0	i=1
M_1	0	s_1	$s_2/\{1,2\}$	$s_1/3$
	0	s_2	$s_3/4$	$s_2/\{5,6\}$
	1	s_3	$s_1/\{7,8\}$	$s_3/9$
M_2	0	s_4	$s_5/\{10,11\}$	$s_6/12$
	1	s_5	$s_6/4$	$s_5/\{14,12\}$
	0	s_6	$s_4/\{7,8\}$	$s_5/9$
...	...	...	...	...

One way to achieve security fusion is to establish a consensus among the response pattern of multiple nodes. Take for example the case of sensor sprays, in which the system wants to make a decision on the detectability of some chemical. This decision can be represented as a binary bit (0/1), and the state machine transitions may be used to communicate this information (detection/non-detection).

For a reliable and secure assessment, the backend system collectively authenticates a response pattern according to a consensus criterion, which could be such that 80% of the nodes should get authenticated. Only if the consensus criterion is met, does the backend system apply a majority logic-a fusion algorithm– on the responses from the nodes to make a decision. If the consensus criterion is not met, then the system, acting as a decision maker, discards all the responses. The criterion is a configurable parameter that sets a threshold on the percentage of responses needed to pass the authentication before a decision is made regarding the transmitted data (which can be either 0 or 1). Unlike distributed approaches [11] [12] [30], the agreement here is centralized.

Aggregating multiple values into a single metric is one example of fusion. Given *N* nodes, an attacker needs to compromise at least *N/2* state machines to influence the decision of the system. While this factor is correlated with high number of nodes, the security of the system is not simply a linear correlation. For the state machines produced in our design, we can show that the security of the system is amplified. But to achieve this, we need to specify exactly how the state machines are generated to guarantee the required properties.

C. State Machine Selection

Because there are many possible configurations for a state machine, a stringent criteria is needed to select state machines with strong security properties. In this section, we describe a criteria for selecting state machines in order to provide protection for the overall architecture. This selection can be implemented as a tool which heuristically explores the universal set of state machines given the number of states n and the number of pseudonyms per state k. The selection is based on the following factors:

1) *State reachability*: State reachability refers to complete reachability in which every state would have a path to every other state through one or more transitions. Any state machine with unreachable states has reduced security complexity. Therefore, in the selection of state machines we should disregard configurations with unreachable states or with potential of being unreachable after a number of transitions.
2) *State complexity*: The generated state machine is stored as a deterministic automaton in the node. Using statistical properties, an attacker can only simulate a non-deterministic version of the machine. Converting a non-deterministic machine to deterministic may lead to exponential blow up in the number of states. However, this explosion does not occur for all non-deterministic machines. Therefore, choosing structures that lead to explosion in the state space is the key for increasing the complexity. In [25], Sutner proposed a class of non-deterministic machines that satisfy the desired property.
3) *Pseudonym randomness*: Pseudonyms need to be cryptographically secure so that an adversary does not predict the values. Since pseudonyms are generated prior to assignment, they can be pre-computed using crypto classes of algorithms. These algorithms can either be stream or block ciphers. Examples of current crypto implementations include Blum Blum Shub [6], ANSI X9.17 [8], CryptGenRandom [2], Yarrow [18], and Fortuna [13].
4) *Pattern randomness*: Pattern randomness highly depends on the selection of the pseudonyms in every state. We propose to use different randomness tests to choose state machines with higher output distribution. Using a black box approach, a random input stream of 1's and 0's is generated to evaluate the distribution of the pseudonym outputs by the state machine. A basic statistical test is to analyze the frequency of the outputs to measure the distributed equiprobability. Other tests based on chi-square [9] including serial and poker tests may be used. These tests evaluate the frequency for a sequence of numbers of some length, e.g. sequence of every five numbers at a time.

V. Security Fusion Based On Compositions

In this section, we introduce the second fusion method based on state machine compositions. Within compositions, there are different types of interconnections including cross products, machine chains, and feedback. Given these basic forms, we demonstrate how compositions can be used to generate a system-wide output for security. First, we illustrate the use of cross products to combine the response pattern of state machines stored at the individual nodes. Afterwards, we present the composition models and the authentication procedure. Finally, we describe a mechansim for complexity reduction based on state machine chaining.

A. Notations

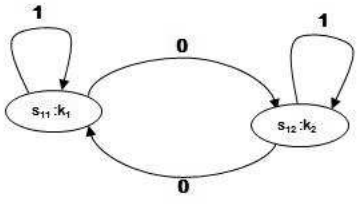
In the following, we introduce the notations/terminologies of this section (Table IV).

TABLE IV
NOTATIONS

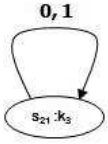
Symbol	Description
M_i	Moore state machine, derived from the nodes.
P	Product machine, constructed using a composition of multiple Moore machines.
C_i	Machine chain component.
s_{ij}	State *j* in some Moore machine M_i.
x_{ij}	State *j* in some machine chain component C_i.
p_j	State *j* in the product machine P
$(s_{1j_1}, s_{2j_2}, ..,)$	Composition state representation of the product machine state p_j
$k_1, k_2, k_3, ..$	Node responses generated and assigned uniquely to the Moore machines.
$s_{ij} : k$	An assignment of a node response k to state s_{ij}

B. State Machine Examples

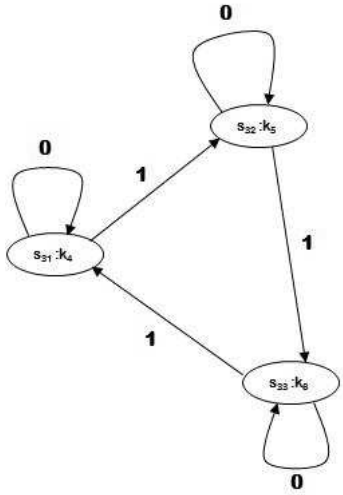
We also depict three state machines M_1, M_2, and M_3 for our examples.



a) Machine M_1



b) Machine M_2



c) Machine M_3

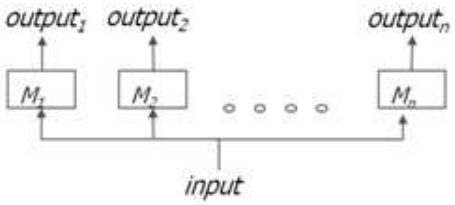
Fig. 3. Machine examples

C. Cross product

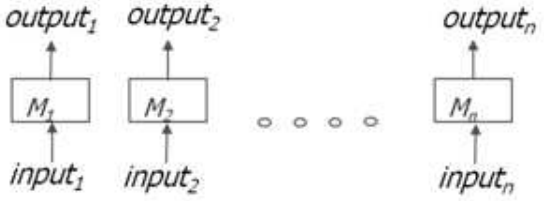
A cross product machine [14] [15] is a state machine that simulates concurrent execution of multiple machines. In other words, each state in the cross product machine represents a configuration of states for a group of machines. Two cross product models simulate parallel execution of state machines:

- *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 logical diagram for both models is depicted in Figure 4. 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.



a) Restricted cross product



b) Full cross product

Fig. 4. Cross product models

Example: Consider the three Moore machines M_1, M_2 and M_3 shown in Figures 3(a), (b), and (c). The restricted and full cross products for the three machines are depicted in Figures 6(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 6, 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 *0*, then M_1 moves to s_{12} and outputs k_2, 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.

D. 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 3.

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 5). In this example, assume the reader input is *0*, and also assume that we are going to

TABLE V
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_{12}, s_{21}, s_{32})	$k_2k_3k_5$
p_6	(s_{12}, s_{21}, s_{33})	$k_2k_3k_6$

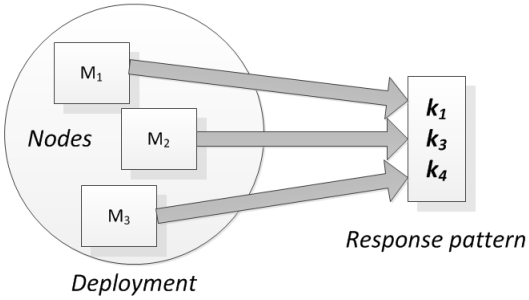


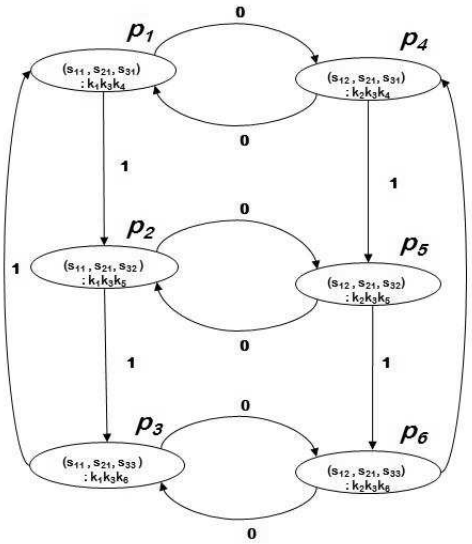
Fig. 5. Response pattern example

apply the restricted cross product, as illustrated in Figure 6(a). The corresponding responses for this configuration are k_2 for M_1, k_3 for M_2, k_4 for M_3. Collectively, the reader obtains $k_2k_3k_4$ as a response pattern (Figure 5). Table V illustrates the mappings for every response pattern to a product state in the system. As shown from the table, the pattern $k_2k_3k_4$ corresponds to state p_4 in the product machine, and the initial states (s_{11}, s_{21}, s_{31}) correspond to p_1. Accordingly, the system accepts the response pattern since $(p_1, 0) \rightarrow p_4$ is a valid transition in the product machine. On the contrary, consider a scenario in which M_3 is replaced with a malicious node that outputs k_5 instead of k_4. In this case, the system obtains $k_2k_3k_5$ as a response pattern instead of $k_2k_3k_4$. Referring to the table, we deduce that the new response pattern corresponds to p_5 and since the transition $(p_1, 0) \rightarrow p_5$ is not a valid transition, the response pattern is rejected by the system.

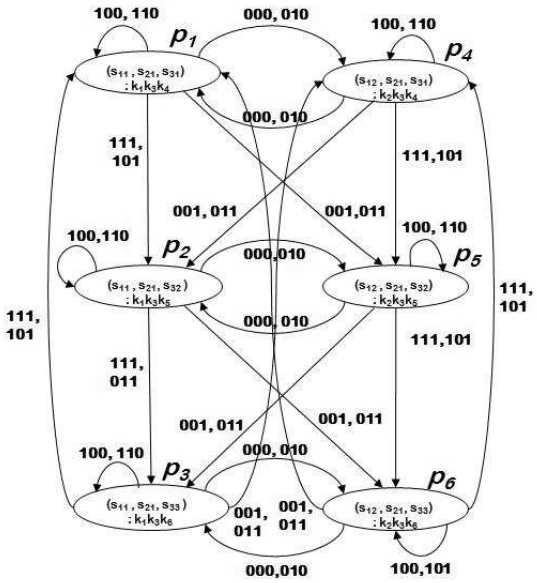
E. Complexity Reduction Using Machine Chaining

We describe a transformation called *machine chaining* to reduce the state space incurred from the cross product construction. State machine chaining [14] [15] 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 interconnection, every state machine is treated as a component such that the state of one component depends on the state of the previous component. 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*.

The theory of machine chaining has been around for decades 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. Here 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 [15] such that the components of this chain are "algebraically" simpler than the original state machine.



a) Restricted product *P*



b) Full product *P*

Fig. 6. Cross product examples

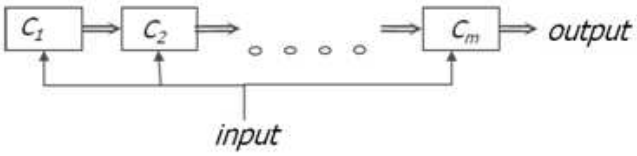
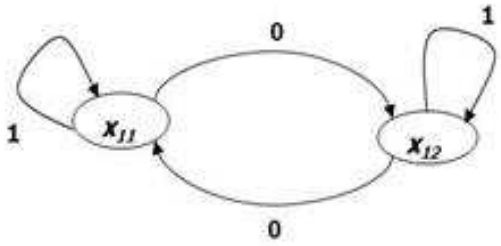
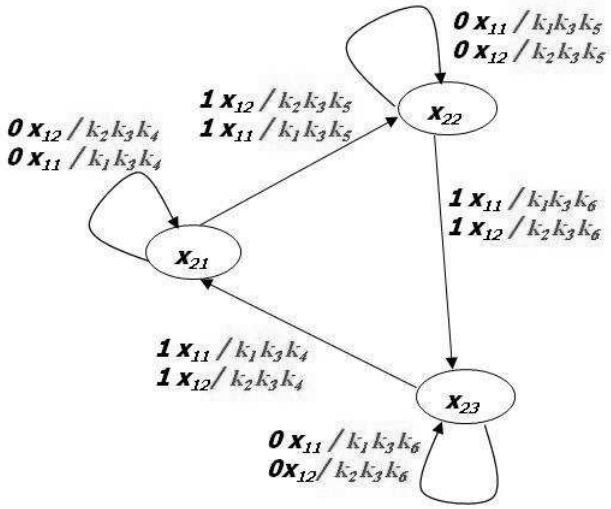


Fig. 7. Machine chaining



a) Independent component C_1



b) Dependent component C_2

Fig. 8. Machine chain components

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 6(a). Depicted in Figure 8, 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 8(b)). Referring to Figure 8(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 VI shows this correspondence.

TABLE VI
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}

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 9). 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 obtains the state machines for all the nodes. The state machines $(M_1, M_2 \dots M_n)$ are initially stored in the system.
- Step 2: Next, we construct a cross product P from the machines $(M_1, M_2 \dots M_n)$. 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 $(C_1, C_2, \dots, C_m)$. Various chaining algorithms perform these conversions, as described by [14].
- 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. For efficiency, the truncation is done in the same pass of factoring P into chain components.

Authentication through chaining: To illustrate the concept, we continue with our example from Figure 5 and the three state machines M_1, M_2 and M_3 shown previously in Figures 3(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 8(a) and (b). According to Tables V and VI, 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

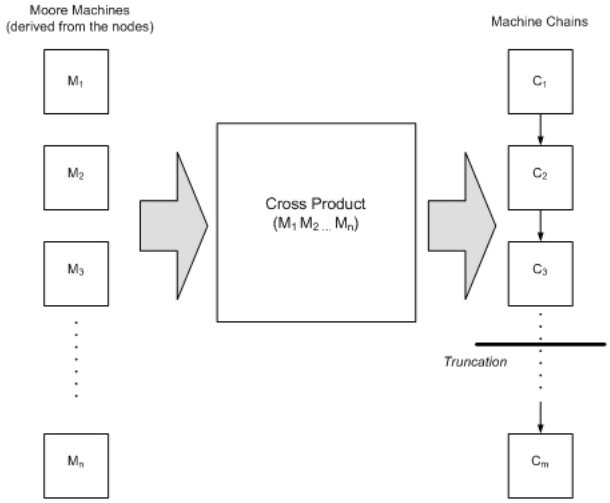


Fig. 9. Chain transformation and truncation

transition rules $(x_{11}, 1) \rightarrow x_{11}$ and $(x_{23}, 1x_{11}) \rightarrow x_{21}$, 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.

VI. Security Analysis

In this section, we evaluate the state machine approach based on majority. First, we analyze how state machines can be used to build a fusion property. Second, we explore the design space of the state machines and the complexity of an exhaustive key search attack. Third, we investigate the packet overhead incurred for malicious reads. The details of the analysis are presented in the subsections below. The analysis parameters are shown in Table VII.

TABLE VII
Analysis Parameters

Parameter	Description
n	Number of states in a state machine
k	Disjoint set of pseudonyms per state
p	Number of pseudonyms to represent transition "0"
q	Number of pseudonyms to represent transition "1"

A. Security Fusion Based On Majority

In the earlier sections, we described how state machines do not reveal the actual state mappings; instead, an attacker can only collect the pseudonyms assigned to the states. The pseudonyms are used to hide the internal representation of the machine, so if an attacker is going to reconstruct the state machine, he will need to collect successive pseudonyms by observing the output pattern.

State machine complexity: One simple way an attacker has of reconstructing the state machine is to treat every pseudonym as a state value. Based on that, the state machine derived by the attacker is non-deterministic (NFA) with nk states and nk^2 transitions for every node, where n is the number of states in the encoded machine and k is the number of pseudonyms assigned to each state. Though an attacker is able to model the readout patterns, the state machine constructed by the attacker is larger in size (with nk states and nk^2 transitions for every node). For the 3-state machine example shown in Figure 2, the state machine constructed by the attacker consists of 9 states and 27 transitions.

State machine minimization: It can be shown from automata theory that the state machine derived by the attacker is equivalent in behavior to the actual state machine. With a large number of nodes, the complexity for storing the NFAs is formidable for the attacker because it requires vast amount of memory and processing capabilities at his disposal. Therefore, in order to scale the attack, the attacker needs to resort to some sort of a minimization algorithm in order to reduce the size of the state machines produced by the NFAs. One of the classical algorithms to minimize these state machines is an algorithm proposed by Hopcroft's [22]. Hopcroft's Algorithm is a minimization procedure which transforms a given state machine into an equivalent state machine with minimum number of states.

Based on Hopcroft's Algorithm, it takes $mlog(m)$ steps to minimize a deterministic finite state machine (DFA) with m-states. Hence, if the state machine derived by the attacker was deterministic, it would take $(nk)log(nk)$ steps to minimize it. However since the state machine derived by the attacker is non-deterministic, we cannot apply the algorithm directly. Instead, the attacker needs to convert the NFA to a DFA first before applying the algorithm. Unfortunately for the attacker, there exists some NFA which leads to a state blowup when converted to a DFA and whose minimal equivalence has $\geq 2^{m-1}$ states. This is a long-known result in automata theory [22] [25]. Because of the state blowup in the conversion, the overall problem becomes *NP-hard* whenever you do not start with a DFA. Not all NFAs will produce an exponential blowup, and hence, a selection criteria is needed to guarantee this property.

The number of states at each node is small by design to satisfy the resource constraints. An attacker may be able to exploit one or few nodes, but a large number of nodes would be exponentially complex to reduce. The question as to how many nodes are needed to have a practical sense of security will depend on the underlying assumptions on the resources available to the attacker and the size of the individual state machines.

B. Exhaustive Key Search

In a brute-force key search, an attacker exhaustively searches all state machine configurations to capture the node behavior. In this section, we explore the solution space an attacker has to exhaust given he knows the pseudonyms in advance. Using the results from combination theory, let us

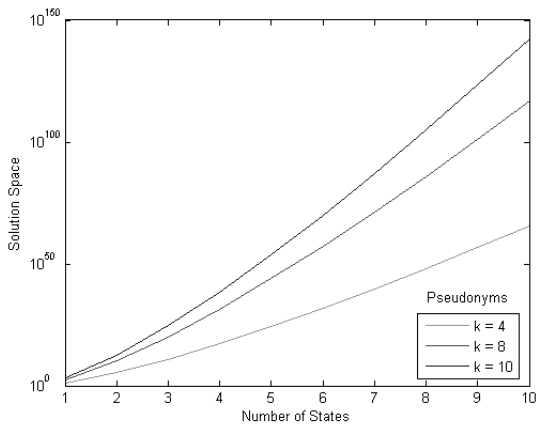


Fig. 10. Solution space analysis

step through several observations to capture the total number of state machines that can be possibly generated. Refer to Table VII for the analysis parameters.

Observation 1: With two outgoing transitions on each state, the number of state machines that could be generated is n^{2n}.

Observation 2: The number of ways of partitioning a set of *n* objects into *r* cells with n_1 elements in the first cell, n_2 elements in the second, and so forth, is:

$$\frac{n!}{n_1!n_2!n_3!...n_r!}$$
where $n_1 + n_2 + ... + n_r = n$

Observation 3: If each state is assigned a set of k pseudonyms, then the number of possible ways that we can assign pseudonyms to a state (using Observation 2) is:

$$\sum_{p=1}^{k-1} \frac{k!}{p!(k-p)!}$$

Observation 4: The number of ways of partitioning a set of nk pseudonyms into n states with k elements in each state is:

$$\frac{nk!}{(k!)^n}$$

Observation 5: The total number of possible state machines that could be generated (using Observation 1, Observation 3 and Observation 4) is:

$$\left[n^{2n}\right] \left[\sum_{p=1}^{k-1} \frac{k!}{p!(k-p)!}\right] \left[\frac{nk!}{(k!)^n}\right]$$

Figure 10 is a plot of the last observation which shows the solution space for generating every possible state machine. We vary the number of states (n) from 1 to 10 and plotted for different values of pseudonyms ($k = 4$, $k = 8$, and $k = 10$). From the figure, we observe an exponential expansion in the key space by increasing the key size in two dimensions: the number of states n, and number of pseudonyms k. For example, if we choose a configuration with 5 states and 8 pseudonyms per state ($n = 5$, $k = 8$), we obtain a solution space of 7.9052×10^{43} which is equivalent to brute-forcing a 146-bit key string.

C. Malicious Reads

With a rogue reader in possession, an attacker may participate in the protocol by making an attempt to read out from the nodes and then launch a replay attack at a later time. Since there is no mechanism in the proposed architecture to authenticate a reader, anybody with a reader can query a node. By interrogating repeatedly, the attacker may collect packets to determine the state machine.

Collection probability: In order to get all the nk pseudonyms assigned to a state machine, the attacker would have to collect a significant number of packets. Assuming the readout pattern is pseudorandom, the following equation provides an estimate of the number of packets that need to be read based on 90% probability.

$$F(r) = \frac{\sum_{p=0}^{nk} (-1)^p C(nk,p) * (nk-p)^r}{(nk)^r} = 0.9$$

(The coefficients in front of the powers come from Pascal's triangle and r represents the number of packets read.)

This probability is a measure of the likelihood for collecting all the nk pseudonyms associated with the state-machine. However, this is insufficient to derive the secret because the attacker will have to detect the transitions of the state machine. The number of transitions the attacker will have to detect will be:

$$F(r) = \frac{\sum_{p=0}^{nk^2} (-1)^p C(nk^2,p) * (nk^2-p)^r}{(nk^2)^r} = 0.9$$

Here the analysis in the above two equations are based on a collection probability, which is the likelihood an attacker successfully determines the state machine by collecting a series of packets.

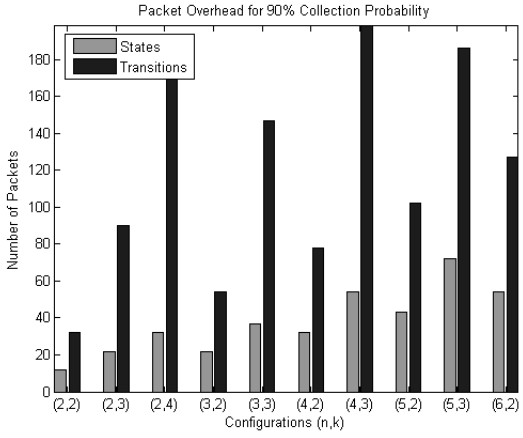


Fig. 11. Collection probability of pseudonyms

The transitions are detected by analyzing successive readouts. To quantify the average packet overhead, we show the

plot for the number of packets to detect all pseudonyms and transitions for different state machine configurations (Figure 11). The light-shaded bar represents the pseudonyms and the dark-shaded bar represents the transitions. As shown in the figure, if we have a configuration of two states and three pseudonyms per state (i.e. $n = 2$ and $k = 3$), then the attacker can be 90% sure he can determine all the nk or 6 pseudonyms associated with the state machine by reading at least 23 packets. As we increase k, the packet overhead to detect all transitions increases quadratically because an attacker has to analyze successive packets.

VII. Experimental Evaluation

This section considers evaluation of the security fusion architecture based on compositions. Since exact analytical solutions are not available, we use simulations to assess the system coverage-overhead of applying the conversions into chain machines. We generate a set of random state machines, apply the cross product, and then transform the product into chains.

A. Simulation Framework

Our simulator was developed using Java JDK1.6 and JFLAP7.0 [3] [21] Jar library which provides basic data structures and algorithms for finite state machine analysis. The simulator consists of:

- Generation of elemental state machines at each node.
- Selection of state machines of strongly connected components using Tarjan's algorithm [26].
- Compositor which constructs the product state machine using full and restricted input combinations. The compositor is optimized for reachable set of states.
- Chain converter that uses state cover analysis.
- Environment to create and simulate node readout, attacker intrusion, and authentication.

Several chaining approaches [14] can be used to transform a state machine into machine chain. In our simulator, we used Permutation-Reset (PR) machines for factoring. PR-machines are special structures which satisfy the following property: All transitions either permute to all states or reset to a particular state. These structures provide a systematic way of chaining in our simulations and are useful because they guarantee to factor out any state machine.

To scale the simulator for large number of nodes, we applied compositions in a hierarchical fashion by taking the composite of small groups of nodes and then combining the results into a single chain machine. In this way, the properties of individual state machines are preserved at the same time the simulator does not have to deal with state expansion from the cross product of all of the nodes.

The simulations were run on top of Intel Xeon@3.4GHz Unix servers. Statistics were taken for multiple sets of 10 runs and various command line options can be used to configure each run including number of nodes, chain length, number of attackers, and intrusion frequency.

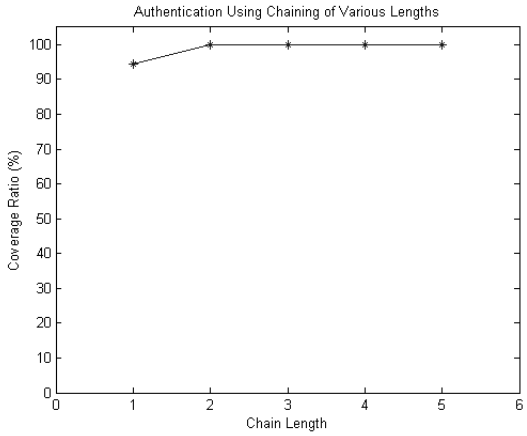


Fig. 12. Coverage Analysis of Chain Components

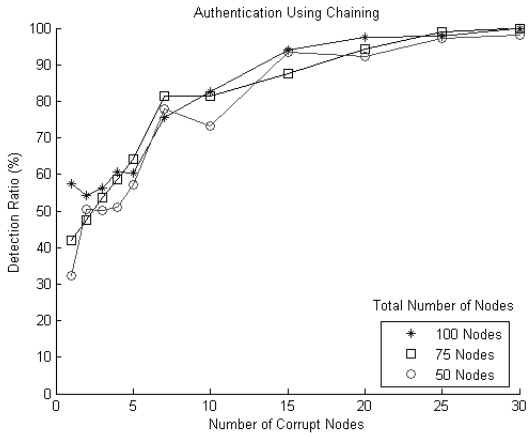


Fig. 13. Detectability of Chain Machines

B. Simulation Results

In order to evaluate the efficiency of the system, we need to know whether the chain length can be reduced without much coverage loss. Furthermore, we need to analyze the overhead and detectability of the system. For that, we define the following metrics.

- *Coverage Ratio*: Defines a proportion of detected intrusions on the system. This proportion helps us set the truncation point of the chain machine to achieve an optimal coverage-overhead tradeoff. We analyze the detection factor of chain components to evaluate how much does each component add to the system.
- *Detection Ratio*: Defines the percentage of intrusions detected. This metric is an overall indication of detectability and is calculated based upon the number of successful intrusions detected over the total number of intrusion attempts.

The coverage ratio across various chain machine lengths is shown in Figure 12. As can be seen, nearly 99 percent of detected intrusions were determined by the first 5 components in the chain. This very high coverage ratio suggests a cost-benefit advantage for truncating the chain to a small number of components, without compromising much of the detectability

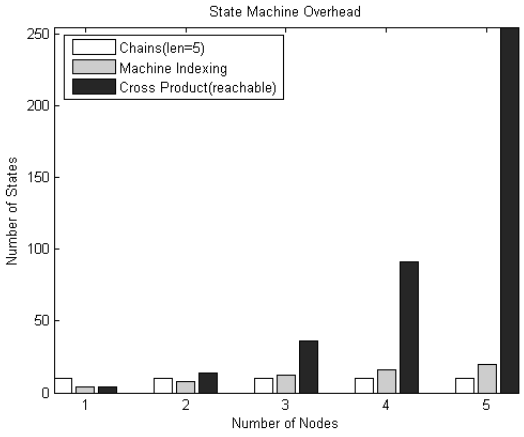


Fig. 14. State Machine Overhead

of the system. Using two-state PR-machines, a truncated chain enables early detection for most attacks since the components are exponentially weighted, with the first component having the largest coverage (about 94 percent).

Figure 13 gives the detection ratio of the authentication system using chain machines of length 2. The simulation size is 50, 75, and 100 nodes respectively. As shown, the sharp rise in the plots demonstrates aggressive reaction to random injects of attacks. The response pattern produced by these injects do not follow any stateful behavior. Consequently, they can be easily found by checking incorrect state transitions.

The number of states for a truncated chain machine, as shown in Figure 14, is invariant of the number of nodes while the number of states for a cross product is exponential (even for the reachable set.) Since chain machines can be truncated at any length, they can provide a significant reduction over linear indexing of state machines.

VIII. Conclusion

In this paper, we have described security fusion architecture based on finite state machines. Since standard cryptographic solutions are likely to be impractical for such an infrastructure, our architecture employs lightweight techniques. Two fusion methods were presented in this architecture. The first method is based on majority logic of state machines that use obfuscation to increase complexity. The second method is based on state machine compositions.

Security analysis of the proposed methodologies presented interesting amplification properties of state machines. Experimental evaluation presented the design tradeoffs. Certainly, optimizations are needed to overcome the ongoing challenges to the deployment of secure systems in resource-constrained environments. As for future work, we plan to investigate other proposals of security fusion in order to achieve more scalable and secure architectures.

References

[1] "EPC Global. http://www.epcglobalinc.org/home/." [Online]. Available: http://www.epcglobalinc.org/home/

[2] "Microsoft CryptGenRandom Function." [Online]. Available: http://msdn.microsoft.com/en-us/library/aa379942(VS.85).aspx

[3] "JFLAP Version 7.0," August 2009. [Online]. Available: http://www.cs.duke.edu/csed/jflap/

[4] O. Al-Ibrahim and S.Nair, "Security Fusion Based on State Machine Compositions," in *IEEE Symposium on Computational Intelligence in Cyber Security Organized in IEEE Symposium Series in Computational Intelligence(SSCI 2011)*, April 2011.

[5] B. Balasubramanian, V. Ogale, and V. Garg, *Fault Tolerance in Finite State Machines Using Fusion*. Springer, 2008, pp. 124–134.

[6] L. Blum, M. Blum, and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," *SIAM Journal on Computing*, vol. 15, p. 364383, 1986.

[7] L. Bolotnyy, S. Krize, and G. Robins, "The Practicality of Multi-Tag RFID Systems." In Proc. International Workshop on RFID Technology - Concepts, Applications, Challenges (IWRT), 2007.

[8] *ANSI X9.17 Pseudo Random Number Generator (RNG)*, Cadence Design Systems Std. I-IPA01-0087-USR, Rev. 6, March 2008.

[9] L. Chernoff, "The Use of Maximum Likelihood Estimates in X2 Tests for Goodness-of-Fit," *The Annals of Mathematical Statistics*, vol. 25, pp. 579–586, 1954.

[10] Crossbow, "Crossbow. http://www.xbow.com/." [Online]. Available: http://www.xbow.com/

[11] G. Cybenko and J. Guofei, "Developing a Distributed System for Infrastructure Protection," *IT Professional*, vol. 2, no. 4, pp. 17–23, 2000.

[12] L. Escenauer and V. Gilgor, "A Key-Management Scheme for Distributed Sensor Networks." Proceedings of the 9th ACM Conference on Computer and Communication Security, 2002, pp. 41–47.

[13] N. Ferguson and B. Schneier, *Practical Cryptography*. Wiley publishing, 2003.

[14] F. Hennie, *Finite-State Models For Logical Machines*, 4th ed. New York, USA: John Wiley & Sons, 1968.

[15] W. Holcombe, *Algebraic Automata Theory*. Cambridge University Press, 1982.

[16] H. Kang, S. Hong, Y. Song, M. Sung, B. Choi, J. Chung, and J. Lee, "High Security FeRAM-Based EPC C1G2 UHF (860 MHz-960 MHz) Passive RFID Tag Chip," *ETRI Journal*, vol. 30, no. 6, pp. 826–832, 2008.

[17] C. Karlof, N. Sastry, and D. Wagner, "TinySec: A Link Layer Security Architecture for Sensor Networks." Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys), 2004.

[18] J. Kelsey, B. Schneier, and N. Ferguson, "Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator." Sixth Annual Workshop on Selected Areas in Cryptography, 1999.

[19] P. Maurer, "Logic Simulation Using Networks of State Machines." Paris, France,: Proceedings of the conference on Design, automation and test in Europe, 2000, pp. 674–678.

[20] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar, "SPINS: Security Protocols For Sensor Networks." Proceedings of Seventh Annual International Conference on Mobile Computing and Networkings, 2001.

[21] S. Rodger and T. Finley, *JFLAP - An Interactive Formal Languages and Automata Package*. Jones and Bartlett, 2006, iSBN 0763738344.

[22] S. Skiena, *The Algorithm Design Manual*. New York,: Springer-Verlag, 1998.

[23] S. S.Nair and O.Al-Ibrahim, "Security Architecture for Resource-Limited Environments," in *The 7th IEEE International Wireless Communications and Mobile Computing Conference (IEEE IWCMC 2011)*, July 2011.

[24] ——, "Security Fusion: A New Security Architecture for Resource-Constrained Environments," *Proceedings of the 6th USENIX Conference on Hot Topics in Security (HotSec'11)*, 2011.

[25] K. Sutner, "The Size of Power Automata," *Theoretical Computer Science*, vol. 295, no. 1-3, pp. 371–386, 2003.

[26] R. E. Tarjan, "Depth-first search and linear graph algorithms," in *SIAM Journal on Computing (SICOMP)*, vol. 1, no. 2, 1972, pp. 146–160.

[27] C.-H. Wang and C.-W. Huang, "A Collaborative Network Security Platform in P2P Networks." NISS '09. International Conference on, 2009, pp. 1251–1256.

[28] J. Wang, "R&D of Gen2 Tag with Enhanced Security Mechanism," Auto-ID Lab, Tech. Rep., 2009.

[29] D. Wheeler and R. Needham, "TEA, a Tiny Encryption Algorithm." Fast Software Encryption: Second International Workshop, Lecture Notes in Computer Science, 1994, p. 363366.

[30] 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, 2008.