Scalability and Cyber Resilience in Gated Array Blockchain Enabled Devices

This research expands on previous works for usage of blockchain in key negotiation between machines. In addition to providing key negotiation, it expands to enabling value transactions in a network of zero-trust nodes. It further describes attacks in an Internet of Things (IoT) on machines that are constrained in resources similar to those of mobile devices or other types of small components. Attacks are formulated and the method of protection is described in light of architecture vulnerabilities. Each set of key exchanges is modeled and the level of protection and vulnerability is described for five cyber-attacks. The core of the research uses a protocol previously published in a time-dimensioned, randomized set of epochs and uses the reversibility property in the exclusive -or-/-nor- gates (XOR/XNOR). The role of a manager and assistant manager are defined within random dimensioned duration slices or epochs and the time requirements to make the change are measured as network nodes are scaled. An optimized model for election of managers, key generation and propagation into the blockchain network during an epoch change is presented. In that model, number of managers are optimized to yield the smallest completion time for each epoch change.

Keywords: *Cybersecurity, Blockchain, Internet of Things, Consensus, VHDL*

Introduction

This document is a continuation of research on special purpose machines that constitute nodes in a blockchain network operating in zero-trust. The machines are based on the Internet of Things (IoT) formats which are small and prevalent in society today. The dual-mode machines can be either participants or managers in the blockchain network and communicate by storing relevant keys in their blockchain memories. This document explains how those blocks interact and are regenerated in time slices called “epochs” by the “manager” mode of operation of the blockchain node. Randomization of parameters tends to result in greater confusion to potential attackers which decreases the ability to formulate and execute egregious behavior.

As explained in [1], these devices are very sensitive to resource consumption (time, power, memory resources) and that makes the design of solutions for them more challenging. However, in [2] it is explained they are coming of age with newer technologies such as 5G networks (and potentially 6G) and widespread adoption in the medical, automotive, defense and other significant segments of society. These devices are also more prone to attack because of the more simple communication protocols used between them [3].

Many protocols and suites exist for managing permissioned networks, notably the IBM Fabric [4] has been successfully implemented in these situations. The main difference between this approach and those outlined is that these devices do not rely on permissioned networks and establish their own identity and trust by storing keys in their blockchains. In addition, those keys are regenerated again in randomly chosen slices of time to further confuse the attack surface for aggressors.

In this study the following questions are researched:

RQ1: Are the epoch completion times feasible as the networks scale in number of nodes?
RQ2: What if any are the scaling limits for these machines in number of nodes?
RQ3: How can these machines defend themselves against cyber-attacks?

Prior Work

This is continuing research on the application of blockchain principles to gated-array machines. In this area, the focus has been on creation of zero-trust [5] blockchain [6] networks that rely on the XOR and the XNOR gates while randomizing the time-slice to regeneration, the key values and the consensus protocols. Significant results have been achieved in power conservation, time of execution and compaction of results in previous publications [7].

This section provides more insight into previous research in the following topics:

- Zero-Trust Blockchains
- Reliance on Reversible Gate Patterns
- Randomization is critical
- Resource conservation.

A Distributed Network of Zero-Trust Blockchain Machines

This research assumes there are no trust parameters between machines prior to the initialization of communication requests ("zero-trust"). The networked machines have two modes of operation Manager/Assistant Manager and Participant corresponding to their roles in the network at a particular time-slice or "epoch" of operation. An "epoch" is defined as: randomly generated amount of time in the context of the network of blockchain nodes [8]. Based on configuration items, these can be regenerated (with associated keys) within limits as can the key lengths and number of keys. Research so far has used 4 keys which are described below.

The participant mode has previously been studied [7] thus the focus of this work is to test the Manager/Assistant Manager modes of operation in light of the research questions. The operating modes and the four keys typically exchanged are further discussed in the following subsections.

The Manager/Assistant Manager Modes

In the election process, a new Manager is elected by the previous Manager's generation of a random binary array that corresponds as closely to a security key length (explained below) used in previous epochs and comparing those values through an XOR operation to all the security keys available (those security keys are stored as blocks in the blockchain). The resulting array with the most "0"s in the outcome will be deemed the next epoch manager (in case of a tie, the first one evaluated will win). The algorithm is repeated for finding the next closest and that

is deemed to be the Assistant Manager. At least one assistant manager should be elected for the next epoch so as to provide continuity in case of the Manager being absent when the next epoch should begin.

At the start of the new epoch, the Manager will execute the tasks of communicating with the Assistant Manager(s), election of the next Manager/Assistant Manager(s), generation of new epoch and operation keys (described below), additional keys if required, admission/deprecation of new participants, definition of next epoch duration and validation/encryption of new blocks, and propagation of those new blocks.

During the epoch, the Manager will call on the Assistant Manager(s) to communicate the new keys. The Manager will communicate the specific keys to the newly admitted machines and will validate/encrypt and broadcast new blocks for the blockchain. The Assistant Manager(s) will communicate individually with member machines by being assigned to specific epoch of admission and communicating with those machines the new keys that are public to the existing machines. In the event the Manager has not begun their tasks within a given time lapse of the start of a new epoch, the Assistant Manager will assume the duties of the Manager and execute them.

The Manager and the Assistant Manager(s) secret keys for the epoch will have been communicated in the prior epoch to the participant machines in the propagated payload. Once the epoch is concluded, they will return to participant mode for the next epoch (unless they are re-elected, in which case the process above will repeat itself again with those machines).

The Participant Mode

As mentioned above, the machines have dual modes and one of them is the "participant" mode. The scope of this is to dialogue with other machines through binary key exchanges of secrets that are stored in their individual blockchain memories and shared knowledge. The keys employed in dialogues can either be current epoch keys, prior epoch keys or combinations of keys since they are stored in the blockchain are therefore known to all nodes.

Blockchain Blocks/Keys

The keys (in binary valued arrays) are as follows:

- EK: Epoch key (common to all): the key of the epoch being referenced.
- OP: Operation key (common to all): 0 is use XOR, 1 is use XNOR in that epoch.
- SK: Security key (unique to participants): unique in the blockchain
- PK: Private key (unique to participants & known to them and the epoch Manager)

These keys will be used in the following sections of this document.

Communication and Encryption via Reversible Gate Patterns

The 2-way XOR and XNOR truth tables effectively create a three-value data set with 2 inputs and one output. By adding another data point to the set (the operation)

where a "0" corresponds to usage of the XOR and "1" usage of the XNOR to evaluate the inputs this effectively creates a 4 data point set. If this is further extended by creating arrays of 1's and 0's for each value, we can create a 4 column array with the inputs, operation to perform and the output (all in binary forms). The following sections describe the reversibility property of this approach and its usefulness in deriving arrays of keys that are operated by Boolean algebra [9].

The XOR/XNOR Gates and Reversibility

The XOR gate (also known as the "equivalence" gate) is a binary logic comparator that outputs a result of "0" if the two inputs are the same or a 1 if they are different. The XNOR gate reverses the process; if the two are different then it produces a 0 for the output and the inverse if they are the same. A common format for illustration of this concept is the "truth table" which outlines the only possible outcomes for all possible inputs when two binary values are input. The truth tables for the XOR and the XNOR [10] are shown in Figure 1.

Figure 1. *XOR XNOR Truth Tables*

XOR

X (in)	*Y (in)*	*Z (out)*
0	0	1
1	0	0
0	1	0
1	1	1

XNOR

X (in)	*Y (in)*	*Z (out)*
0	0	0
1	0	1
0	1	1
1	1	0

As mentioned in this subsection's introduction, the inclusion of a fourth value in the truth table, the operation to use, effectively transforms the three valued array into a four value array as shown in Figure 2.

Figure 2. *Four Value Array Examples (XOR, XNOR)*

XOR

X (in)	*Y (in)*	*OP*	*Z (out)*
0	0	0	1
1	0	0	0
0	1	0	0
1	1	0	1

XNOR

X (in)	*Y (in)*	*OP*	*Z (out)*
0	0	1	0
1	0	1	1
0	1	1	1
1	1	1	0

Furthermore, if we extend the concept to binary arrays they can be used to derive value when 3 of them are known. This was given in [7] and repeated below for reference:

"For any combination of three binary digits, 1 or 0 when the XOR or the XNOR gate is used to evaluate the logic operation, the fourth digit shall be equal to 0 when any combination of the first three has all the values of 0 or only one of the first digits has the value of 0 else the value of the fourth digit shall be equal to 1."

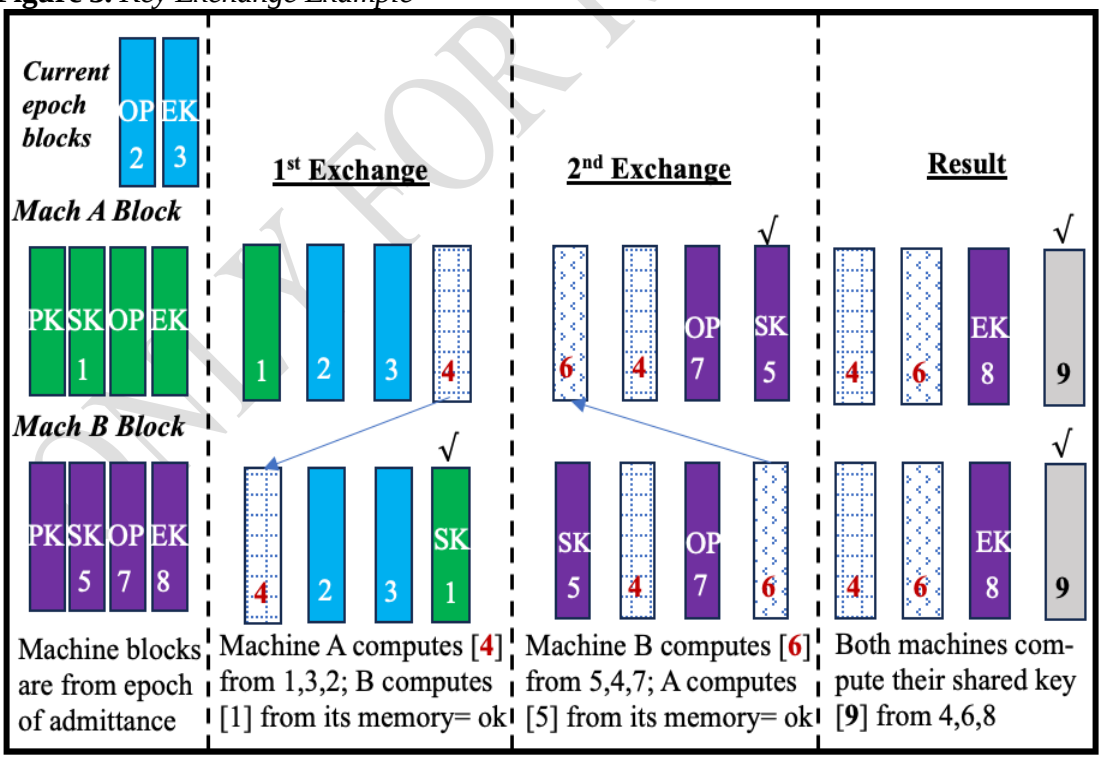
Proof:

$$\{ 0 \oplus 0 \oplus 0 \} = 0 \;\&\&\; \{ 0 \odot 0 \odot 0 \} = 0$$
$$\{ 0 \oplus 1 \oplus 1 \} = 0 \;\&\&\; \{ 0 \odot 1 \odot 1 \} = 0$$
$$\{ 0 \oplus 0 \oplus 1 \} = 1 \;\&\&\; \{ 0 \odot 0 \odot 1 \} = 1$$
$$\{ 1 \oplus 1 \oplus 1 \} = 1 \;\&\&\; \{ 1 \odot 1 \odot 1 \} = 1$$

Key Exchanges and Reversibility of Results

Figure 3 gives a visual example of how key exchanges would work if we had 4 binary arrays interacting with current and prior epoch knowledge. In the current state, all keys are known to machines A and B. Machine A initiates the dialogue request by computing array 4 and sending to machine B. Machine B validates the A's Secret Key 1 by taking its knowledge from the blockchain and computing 1 from 4,2 and 3. It next computes 6 from 5,4 and 7 and sends it to machine A who validates 5 ensuring it is communicating with the right machine. In the final dialogue, they both compute their shared key by taking 4,6 and 8 to derive 9.

Figure 3. *Key Exchange Example*



Randomization as an Attribute

Use of random numbers provides for creation of alternatives that can enhance protection for attack surfaces. Randomization can provide a continuous infinite

distribution that is very challenging to attack since it requires the attacker to reformulate an attack on a moving target. Enhancing solutions with randomization in the following areas can further confuse the attack surface:

- Duration of time between regeneration of keys referenced above
- Length of keys (number of binary digits).
- Number of keys (more can be used in the model).
- Direct election of network managers and assistant managers.
- Voting for managers and assistant managers (not all have to vote).

When randomization is combined with large binary keys it forms a phenomenal protection combination [8]. The aspects of binary key size and traversal of blockchain in order to secure additional required keys are discussed below.

Key Size and Probabilities of Attack Success

As mentioned above, the keys in this document are binary arrays with 1 or 0 in each location. For example if the key (array) is 4 bits long then there are 2^4 number of combinations of 1 or 0s contained in that array (ie: 0000,1000,0100…1111). The probability of guessing without prior knowledge is 1/16 (in decimal notation). With replacement (knowledge that one or more numbers previously guessed are not correct) becomes (in the single case and through exponentiation):

$$P(x) = \frac{1}{N-z} \text{ [11] } P(x) = \frac{1}{[(N1^{y1})*(Nn^{yn})]-z} \text{ [11]}$$

Where:

N : Number available in the Universe or sub-component Universe

z : Number previously drawn

yn : Sub-component Universe size exponent

In the previous example, if a first try did not guess correctly, the next guess would have a $P(x) = \frac{1}{16-1}$ = 1/15 probability of guessing correctly and the number of tries that did not guess correctly would be accumulated until the last guess (if all were incorrect) would have a $P(x) = \frac{1}{16-15} = P(x) = \frac{1}{1}$ probability of guessing correctly (at that point it is a certainty and not a guess).

If the case was expanded to two arrays of the same size, the probability of guessing correctly the first time both combined numbers would be $P(x) = \frac{1}{2^4*2^4}$ or $P(x) = \frac{1}{2^8}$ = expressing the result in decimal is $\frac{1}{256}$; if we had 4 arrays of the same size, the combined probability of success would be $\frac{1}{4096}$. If the first guess was incorrect, the next guess (with replacement) would be $\frac{1}{4096-1}$ and so on, subtracting one more for each failed try. Furthermore, if four keys were used that were 64 bits long, the probability of success would be $P(x) = \frac{1}{2^{256}}$ and $P(x) = \frac{1.158}{10^{77}}$ in decimal notation (1,158 with 74 0's after it). Therefore by implication, adding more keys

and increasing key length further lessens the probability of attack success without knowledge.

Previous Resource Usage Results

Previous work [7] has indicated that exchanges between machines use less than 1/1,000 watt of dynamic power in computation for a 256 bit array key-exchange versus 2.408 watts using a SHA-256 key-exchange using FPGAs. The same study concluded that the speed of execution when compared to Diffie-Hellman's algorithm (also 256 digits) executed in 10.70 CPU seconds vs 0.58 CPU seconds in a digital computer representing a significant differential in favor of the gated algorithm.

The above studies were simulated in peer machines participating in a network exchange dialog.

The Role of Dual-Purpose Machines in the Model

Two key aspects dominate this research; the machine modes and the actual dialogues that happen within those modes. The following sections detail the participant mode communication, the general dialogue model and the manager communications and operations.

Communication in the Participant Mode

The participant mode is the more common mode of operation of the machines. In this mode, the machines interact with each other by exchanging arrays of binary digits or keys. The key lengths can be varied and are stored as blocks in previous or current epochs on the blockchain. In order for the dialogues to be successful, both machines need to know previous and current epoch blocks. These keys are used to execute exchanges which lead them ultimately to exchange something of value between them.

Participant Mode Characteristics

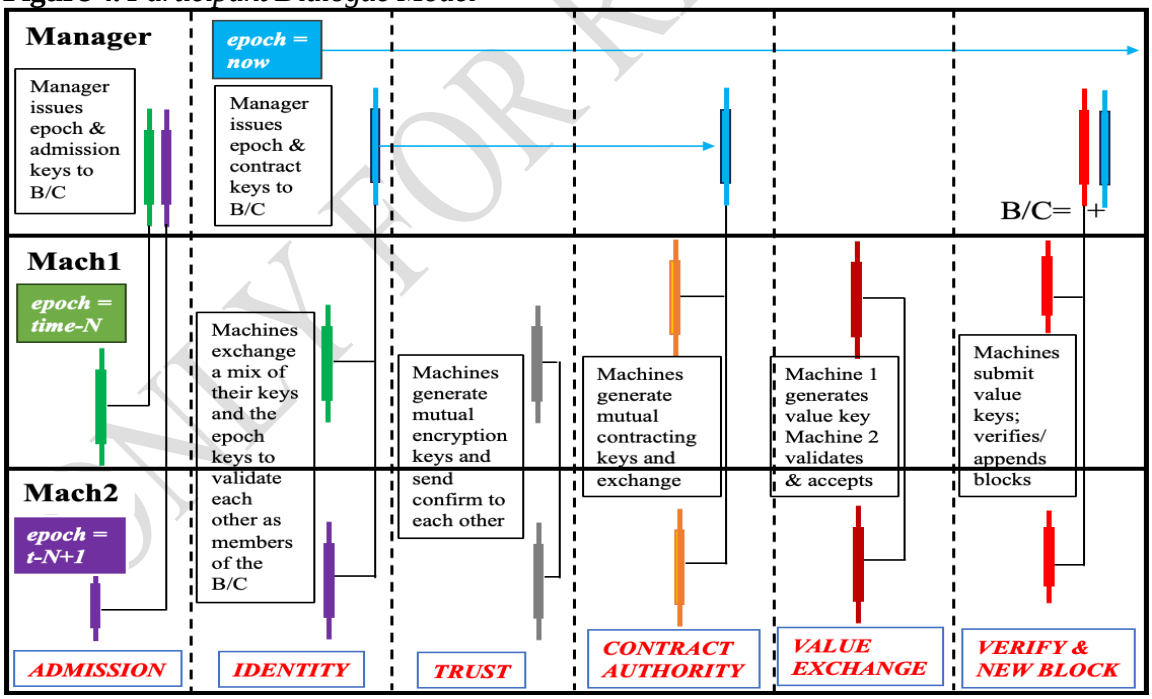
In the participant mode of operation, the machines achieve identity, trust, contract and value exchanges through logic operations using the above mentioned XOR or XNOR gates. Identity is achieved by sharing keys that are derived from components in the blockchain that would not be known to an uninformed participant. Trust is achieved by negotiating a mutual key based on products of blockchain keys that are unique to the two of them. Contract (or the ability to exchange value) is also achieved by another key which is only known to each machine a product of which is communicated using operations with the blocks. Finally, value exchange is achieved by operating the mutual contract keys and notifying the manager to encrypt into the blockchain network. The next section provides an example of that process.

Sample Dialogue Model

In Figure 4, an example of a dialogue model is given and the following events take place:

a. Admission: the two machines have been included in the blockchain by a prior manager in 2 prior epochs. They are given their secure, private and contract keys and also all the current and prior blockchain blocks that are known to their peers in the network.
b. Identity and Trust: one machine initiates the dialogue by sending current epoch keys with their secret key operationalized with the current operation and epoch keys in a fashion described in 2.2.1. Using knowledge of the blockchain, the second machine de-aggregates and confirms the first's secure key and executes a similar operation. Both machines with that knowledge generate a third key that they will use in generation of another contract key in the next phase.
c. Contract Authority and Value Exchange: in this phase, the two machines generate mutual keys with their previous contract keys and use the unique contract keys they have generated to transfer value (one gives value and one receives it).
d. Verify & New Block: in this final phase, the machines independently report their give/receive transactions to the epoch Manager who appends to the blockchain and notifies the network of the new transaction.

Figure 4. *Participant Dialogue Model*



Machine Operations and Communication in the Manager Mode

The manager operation mode is more complex than the participant mode since more operations must be completed for a new epoch to exist and for the appending of blocks. The concept of consensus through manager elections has been explored in the RAFT [12], Intel's PoET [13] and others. These type of algorithms resolve

the common Byzantine General's Problem [14] through an election among the machines. The election algorithm presented here will resolve that selection by the previous manager generating and comparing a random key to the machines' secret keys and finding the machine that is the closest to that generated key.

Once the selection of the future manager has occurred, the manager will proceed to generate either the current or next-future epoch's duration through a random process again and will announce that duration to the next manager. Next steps after this election are: admission and deprecation of machines, generation of operation and epoch keys, appending of blocks to the blockchain, referral of new block to audit function. These key steps are detailed in the following sections.

Manager Tasks

When a new epoch begins, the machine designated as the new epoch manager will switch state from participant to manager mode (this state change will be effected by the passing of time). Next, the manager will elect the following epoch manager and if necessary the number of backup managers that are to be elected (this process of election is discussed later in greater detail). Once the new manager(s) are elected and notified, the manager will execute generation of the following keys:

- Admission keys for each new node:
  - Secret Key (SK) which must be unique in the blockchain
  - Private Key(PK) known to the node and manager
  - Contract Key (CK) known to node and manager
- Deprecation of Secret Keys of machines that are leaving (if any)
- Epoch keys: Epoch Key (EK) and Operation Key (OK)

The manager and assistant managers will propagate the keys by individually notifying each node through the participant dialogue process in figure 4. The nodes will replicate those keys in their local blocks. Finally, the manager will switch to that mode if required in order to append blocks to the blockchain, change states temporarily to enact it's own participant role if necessary or to restart another epoch in the event of a validated attack (see attack sections below). More propagation processes will execute as value exchanges occur.

Election of the Managers and Substitute (Sub) Managers

The election of the managers and any alternate or assistant-managers will be done by random binary number generation. The procedure described below is translated to digital logic hardware definition language (VHDL) below in the experiment section. The procedure is as follows:

:Begin Do

a. Generate Random Number Array
b. Set value to max numbers
c. Traverse Blockchain get next Secret Key (SK)
d. Execute Comparator
e. Execute Accumulator

f. IFF Accumulator < value {value =Accumulator}
g. IFF Next Secret Key {a} else {:End-do}
:End-do

In the above algorithm, items c-g will continue to be executed for the number of alternate or assistant managers required.

Vulnerabilities During Epoch Changes

Times of change in any system increase the risks of operation of that system [5]. The author believes the risk increases during an epoch change due to inherent factors mentioned below:

1. Time to determine new machine elections and notification of those machines.
2. Time required to generate new keys and in the case of Security Keys and guaranteeing uniqueness by traversing the blockchain.
3. Time to propagate new keys throughout the network.
4. Potential for out-of-synchronization between peers (machine thinks it's in a different epoch).

Some of the above are machine dependent, network dependent or both. The most important risk mitigation however, is to shorten the amount when the epoch change is happening. A risk mitigation approach or a large change time might be to have machines communicate in the ending epoch until most or all of the machines have been notified of new epoch changes (overlap of epochs).

Experiments Design

Two forms of experiment were conducted. In the first, the objective is to measure the epoch change when number of nodes are scaled. The second determines the potential attack survivability under five common attack types. The cyber-attacks are defined in more detail (since many variations exist of each), then they are operationalized and finally the vulnerability if any is identified.

Epoch Change Time in Manager Mode of Operation

This section includes an experiment on the model's epoch change time during the Manager's mode of operation. The model is operationalized using gated array and digital logic concepts into hardware language that targets FPGAs [15]. The main objective is to determine the completion time required in order to effect a shift to a new epoch. The VHDL follows patterns defined in [16]. The steps executed and measured in the results section by the toolkit are in figures 5 and 6:

Figure 5. *Election Process VHDL Pseudo-code*

```
// Begin election
: ARRAY1-GEN-RND // generate random array, 64 bits
: ARRAY2-SET-1 // set second array to all 1's
: ARRAY3-SET-1 // set third array to all 1's
: ADDER1-SET-1 // set value to all 1's
: ADDER2-SET-1 // set value to all 1's
    : DO-WHILE MEM-IDX-MORE == // Begin loop to traverse and compare
      : READ-MEM-IDX // Read block in blockchain
      : IFF {SK} DO // if it is a secure key
        : COMPARATOR SK&&ARRAY1→ARRAY3 // compare SK to array 1 eq array 3
        : ADDER ARRAY3→ADDER1 // add up array 3
          : IFF {ADDER1<ADDER2}{ADDER2=ADDER1&&ARRAY1=SK}
            // If adder 1 lt adder 2: adder2 eq adder1 and array 1 equals SK (closest)
      : ENDIF {SK} DO // close logic loop
    : END-WHILE MEM-IDX-MORE // close read loop
// End election
```

The process in figure 5 was scaled according to the following assumptions:

1. Number of managers elected: 1 to 1,200 depending on number of nodes in network (Manager plus assistants/backup).
2. Number of nodes in the blockchain: 1,000 to 100,000.
3. Number of blocks in chain before another Security Key is found: 100 includes notification in epoch plus transactional blocks for exchange of value.
4. Key lengths: 256 and 512 bits.

Figure 6. *Machine Admission Key Generation Process VHDL Pseudo-code*

```
// Begin new keys for admission
: ARRAY1-GEN-RND // generate random array1, 64 bits (CK) Contracting Key
: ARRAY2-GEN-RND // generate random array2, 64 bits (PK) Private Key
  : DO-WHILE !IFF {SK} // while SK is not in blockchain
    : ARRAY3-GEN-RND // generate random array3, 64 bits (SK)
      : DO-WHILE MEM-IDX-MORE == // Begin loop to traverse and compare
        : READ-MEM-IDX // Read block in blockchain
        : IFF {SK==ARRAY3} BREAK // if the secure key exists break
      : END-WHILE MEM-IDX-MORE // close read loop
  : END-WHILE !IFF{SK} // close loop if SK is not in blockchain
: WRITE-IO →ARRAY1&&ARRAY2&&ARRAY3 // communicate keys
// End generate new keys for admission
```

The process in figure 6 was scaled as follows (see Manager election for assumptions above):

1. Number of managers participating in notification: 1 to 1,200.
2. Number of nodes in the blockchain: 1,000 100,000.
3. Number of blocks in chain before another Security Key is found: 100 (corresponding to the number of blocks per node).
4. Key lengths: 256 and 512 bits.

The scaled results for the above 2 experiments are reported in the results section of this document.

Test Infrastructure

The test infrastructure consists of the following:

- CAD: Vivado VHDL Design, Synthesis and Simulation v. 2023.2
- Target FPGA: Xilinx Baysys3 part number XC7A35T-1CPG236C [17]
- OS Environment: MS Windows Tablet, 8MB RAM

Survivability during Attacks

No one really knows how many attacks there are in total in the world today, it is almost impossible to determine, no one agency will accumulate, typify and report them. There are however some that are well known with many variations and five of them will be described in this section. As with most attack patterns, they begin with a small probe volume and are successively scaled in order to progress to a next level (a foothold from which to formulate a new one and proceed with escalation of malfeasance).

The five attacks are described below. The objectives of the attacker are either to exhaust resources (DoS) or to penetrate and begin next stage of attack. The gated array dialogue is presented to indicate the attack progression and repelling if possible.

Brute Force Attack (BF)

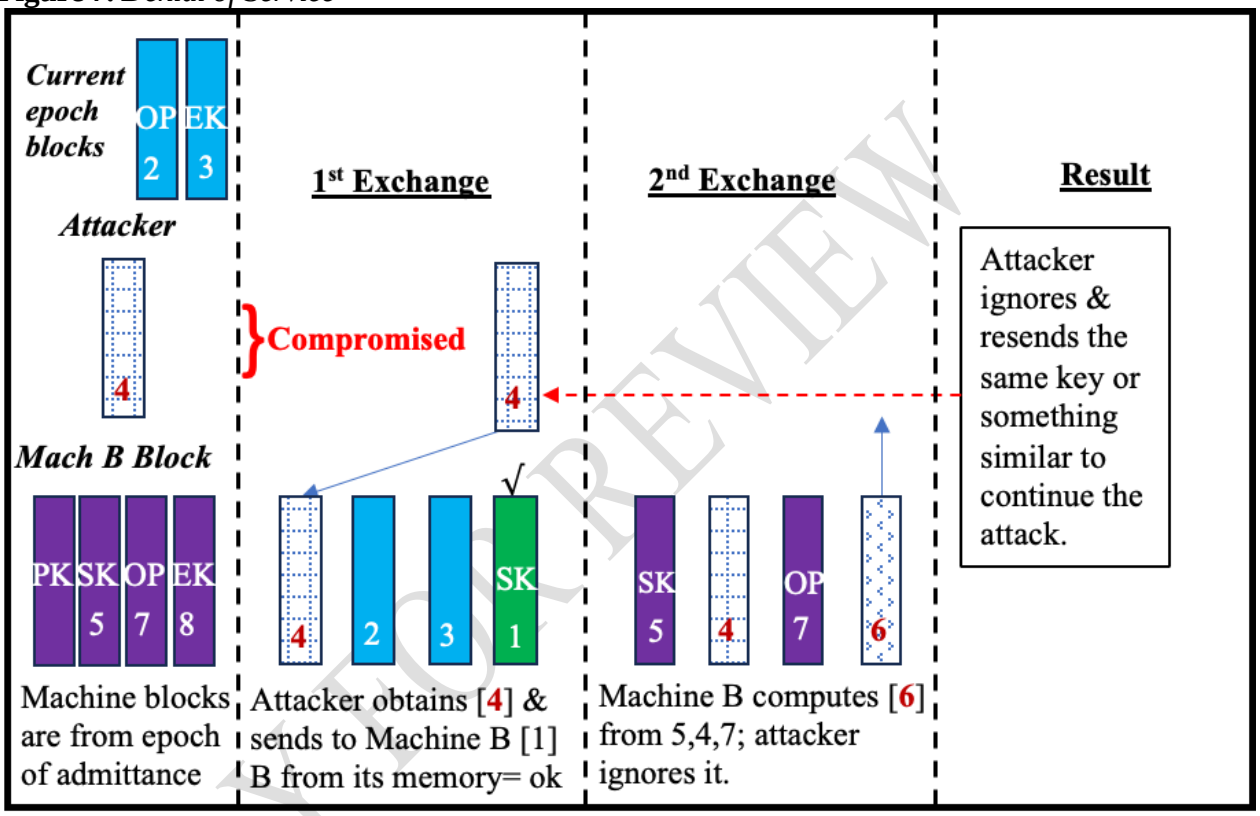
The objective of a Brute Force Attack is to gain an entry point into a target. The method of attack is to generate data and forward that data to the target and try to obtain some feedback as to the method of communication that is typically required to begin a dialogue. As detailed above, the brute force attack due to the nature of the keys has a very low probability of success and is directly proportional to the size of the key. In a first stage, the size of the key is not known to the attacker so it will attack with a variety of sizes and contents. If one of those tries is successful in matching the size, the next stage must start to find a correct key based on the dialogue model.

That second stage is also quite difficult since the correct combination of keys to frame the first valid key must be derived. The probability of that has been shared in 2.4 and until that key has been derived, an attack will not receive feedback as to anything.

Denial of Service (DoS)

The objective of a DoS is to exhaust the resources of the victim by overwhelming it with requests in such a frequency that it will deploy its resources to the fullest in attention of those requests. This outcome denies the resources to legitimate users and effectively makes the node unusable to service its required objectives.

Figure 7. *Denial of Service*



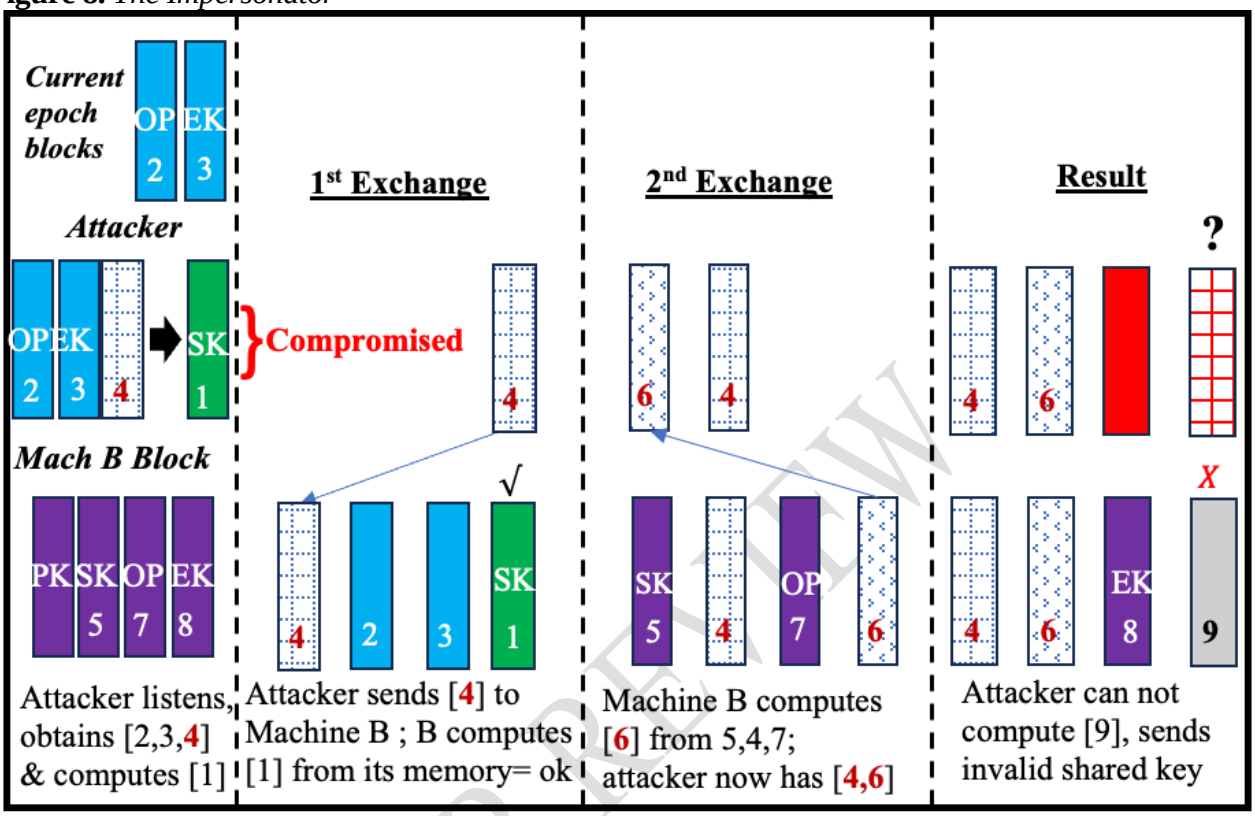
In figure 7, an attacker has obtained limited knowledge of a message that has been sent before through listening, spoofing or some other means. It sends this key to the target machine and the target machine responds after it validates. The response is ignored (since the attacker has no other knowledge) but the attack continues by sending either the same value or other values that are similar.

As the attack is detected (by number of failed tries), the node under attack would request from the manager that another epoch be started as soon as possible.

Credential Simulation (“impersonator”)

The objective of an impersonator attack is to obtain a session into another machine by means of sending valid messages until a “foothold” is achieved in the machine and the next steps of the attack can be progressed.

Figure 8. *The Impersonator*



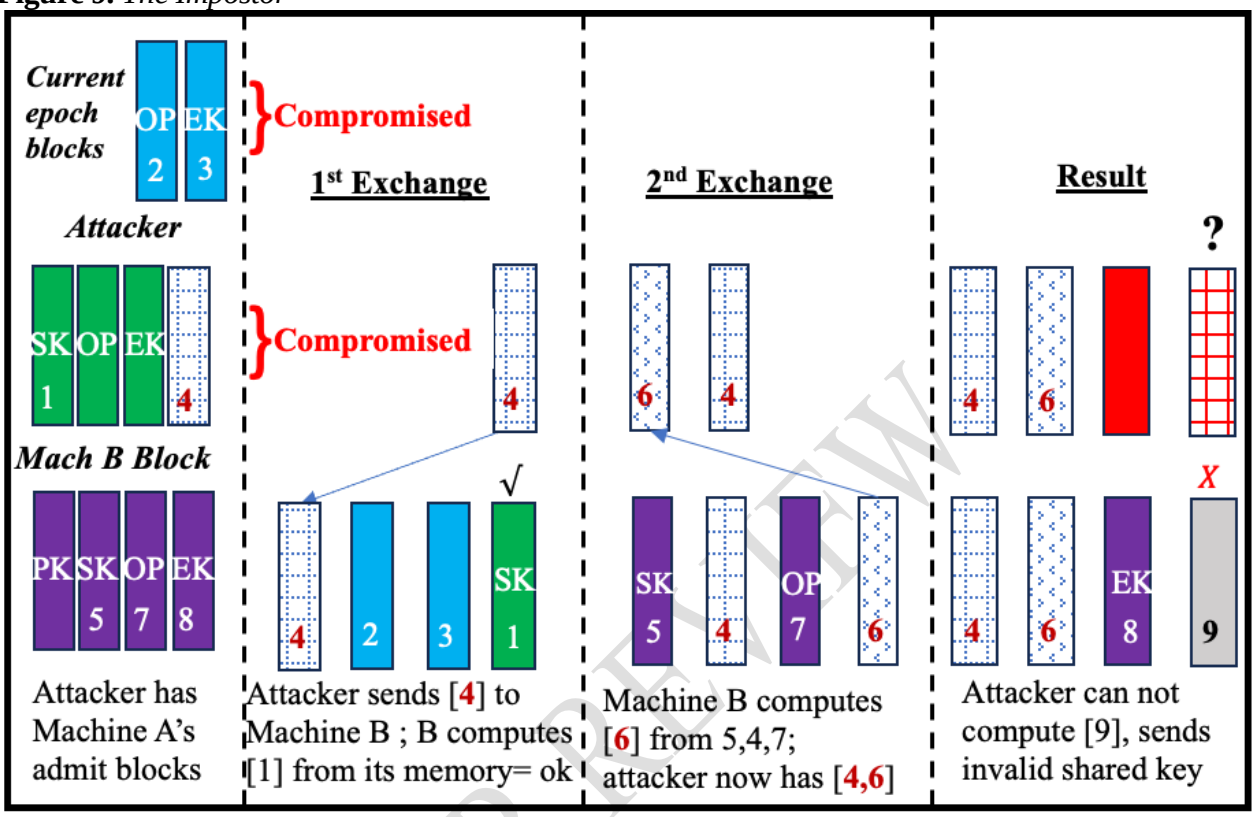
In figure 8, an attacker has obtained knowledge of the key exchange methodology, the current operation and epoch keys and another key that has been used to communicate. It begins the attack with the communication key and does receive a response. This response cannot be progressed since the attacker does not have knowledge of the machine's blockchain and the attack fails when the two machines have different negotiated keys (at step 9 in Figure 8).

In a same manner as the DoS above, repeated failed attempts would elicit the node under attack to request a new epoch be started.

Impostor

The objective of an impostor is to coexist with a valid node and confuse the system by posing as that machine.

Figure 9. *The Impostor*



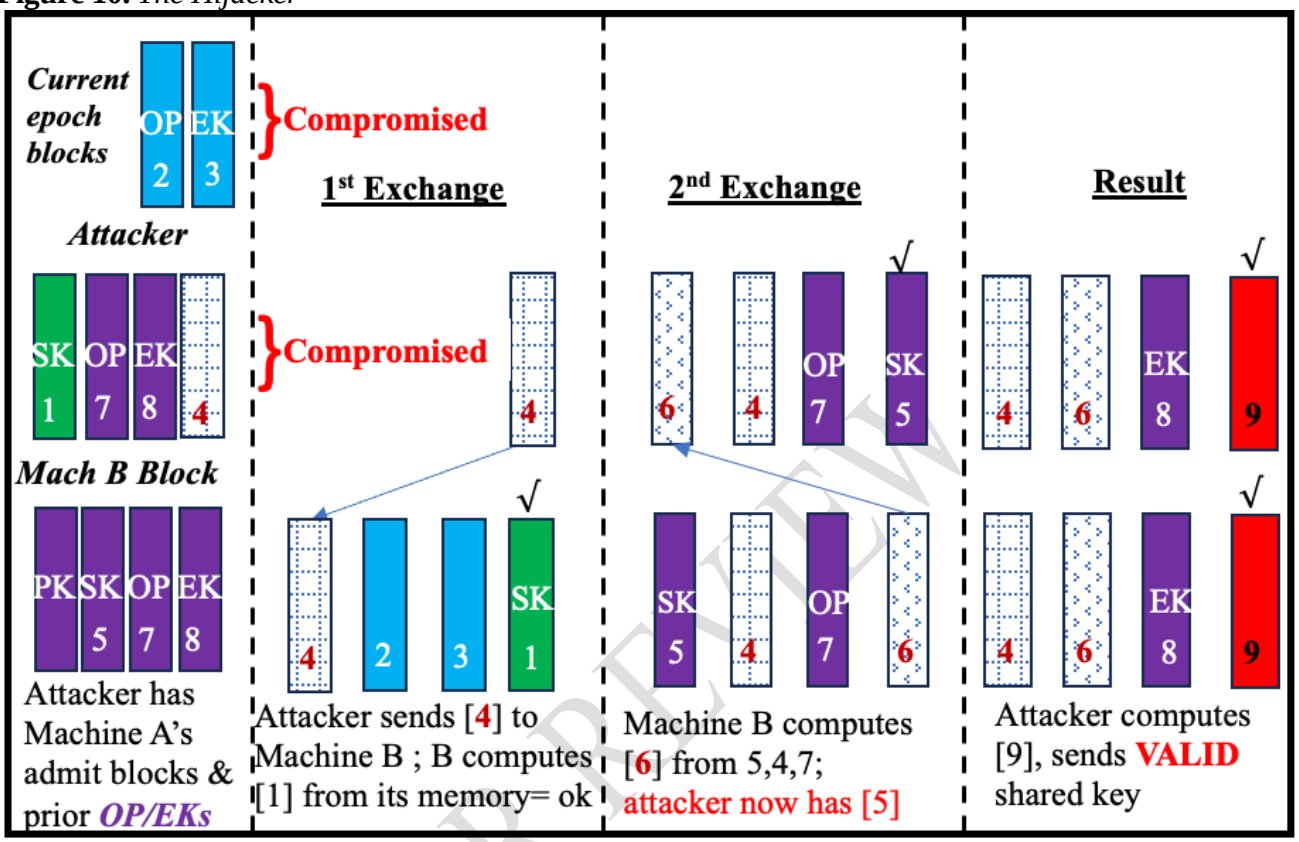
In figure 9, an attacker has obtained knowledge of the key exchange methodology, the current operation and epoch keys, another key that has been used to communicate and a machine's epoch of admission keys. It begins the attack with the communication key and does receive a response. This response also cannot be progressed since the attacker does not have knowledge of the relevant blockchain epoch keys. The attack fails when the two machines have different negotiated keys (in step 9 of Figure 9).

Again, repeated failed attempts would elicit the node under attack to request a new epoch be started.

Machine Control ("hijacker")

The objective of a hijacker is also to execute transactions and exchange of value in the blockchain network to the exclusion of a legitimate machine (hijacking the dialogue to exclude the other). The ultimate goal is to extract value from the network or other nefarious outcome (getting a node to send value to them).

Figure 10. *The Hijacker*



In figure 10, an attacker has obtained knowledge of the key exchange methodology and the necessary components of the blockchain data in order to simulate another machine and extract value. It begins the attack with the communication key and proceeds to negotiate a key (step 9 in Figure 10). The attack potentially will fail in the next stages since the contracting keys are not publicly known in the blockchain rather communicated directly from the managers to the participants upon admission. They are stored at the local level in a contract with the admitting machine's memory. Nevertheless, this level of intrusion is extremely dangerous and might be overcome by trial and error with some knowledge of prior value blockchain transactions. This illustrates that most successful attacks will include insider knowledge [18].

Execution and Results

The Vivado suite in simulation mode for the FPGA mentioned above was used in the election and key generation sub-processes. Two key sizes were selected: 256bits and 512bits and an average occurrence of a security key every 100th block (meaning, every node will produce an average of 100 transactional value blocks). The selection of block frequency is difficult since the Transactionality requirements are defined when the purpose of the blockchain is given (however, one must be

chosen for the model). That size represents a range of exchange of 20 times per year or 100 in one year.

The number of new admissions was simulated to equal the size of the network over 5 years with one epoch occurring every month (plus one key for epoch and one key for operation code). Each security key (SK) was assumed to be unique. The manager node must compute the difference between the generated key and the keys in the blockchain which becomes a factor as the network nodes increase.

VHDL scripts were constructed to simulate the code concepts referenced above and several test scenarios were constructed; one for a "unitary" scope (two manager election, generation of one set of keys for admission of one node and new epoch keys) and more for a multiple-manager/assistant manager election and admission of large amounts of new nodes. Within that scaling environment, the second (multi-participant or "gang" scenario) was further defined into two types of networks; the small with 1,2 and 3 thousand nodes and the large with 10,30,50 and 100 thousand nodes.

Once initial results were received for election and key generation sub-processes, they were combined with previously published results for key dialogue. Please note that the process for the actual computation involved in the exchanges between nodes is minimal compared to the latency in any network (the network is almost always the slowest component). For purposes of notification of keys, it was assumed there would be an initial exchange between machines of 4 keys to establish identity and trust as per the model discussed above and one exchange for the payload of the new epoch and the admitted machines.

It is also important to note that the FPGA used has a limitation of 1,800,000 bits of RAM. This clearly would need to be augmented in a field trial for networks of nodes greater than 3,000 in the attached experiments. FPGAs and IoT devices that can house storage greater than those dimensions do exist and they could be sourced with the same processor configuration as the one used here with reasonably similar results.

Individual Task Component Results

As mentioned above, the results were run on "unitary" and "gang" basis. The unitary results are shown in Table 1. The "gang" results were obtained by first using empirical estimation of a range of high and low participation rates for managers and executed in the vivado suite for each range and network size, those results are shown in Table 2.

Table 1. *Unitary Results*

	Total	Election		Generation		Notification	Epoch
OBS	Nodes	Mgrs.	Time	# Keys	Time	Time	Time
	Key Size: 256 bits						
1	1,000	2	00:09	1	00:09	33:20	33:38
2	2,000	2	00:09	1	00:09	66:40	66:58
3	3,000	2	00:09	1	00:09	4 hours	4 hours
4	10,000	2	00:13	1	00:09	12.5 hrs	12.5 hrs
5	30,000	2	00:17	1	00:09	37.5 hrs	37.5 hrs
6	50,000	2	00:18	1	00:09	125 hrs	125 hrs
7	100,000	2	00:20	1	00:10	250 hrs	251 hrs
	Key Size: 512 bits						
8	1,000	2	00:09	1	00:09	33:20	33:38
9	2,000	2	00:09	1	00:09	66:40	66:58
10	3,000	2	00:09	1	00:09	4 hours	4 hours
11	10,000	2	00:10	1	00:09	12.5 hrs	12.5 hrs
12	30,000	2	00:11	1	00:09	37.5 hrs	37.5 hrs
13	50,000	2	00:13	1	00:09	125 hrs	125 hrs
14	100,000	2	00:17	1	00:17	250 hrs	251 hrs

Table 1 depicts the time observations received when two managers were elected and one machine was admitted. Notification time is calculated as 5 seconds per machine divided by the number of nodes in the network. The process of notification is the same as establishing dialogue up to the trust and identity level plus the communication of the payload (5 seconds per machine as previously noted).

Table 2. *Multiple Managers – Election & Key Generation*

ELECTION		MM's	**Key: 256 b**		**Key: 512 b**		**KEY GENERATION**			**Key: 256 b**		**Key: 512 b**	
Mgrs.	Nodes	Blocks	CPU s	Time	CPU s	Time	Mgrs.	# Keys	Nodes	CPU s	Time	CPU s	Time
10	1,000	0.1	5	00:09	5	00:09	10	19	1,000	3	00:09	4	00:10
20	2,000	0.2	5	00:09	5	00:10	20	35	2,000	4	00:11	4	00:13
30	3,000	0.3	5	00:13	4	00:12	30	52	3,000	4	00:13	6	00:15
100	10,000	1.0	9	00:43	7	00:46	100	169	10,000	5	00:29	6	00:49
200	30,000	3.0	13	02:42	18	04:11	200	502	30,000	17	04:13	34	08:03
420	50,000	5.0	40	10:04	50	13:38	420	835	50,000	20	05:11	42	10:42
1000	100,000	10.0	162	45:22	254	67:03	1000	1,669	100,000	133	28:31	136	38:26
20	1,000	0.1	5	00:09	5	00:09	20	19	1,000	3	00:09	3	00:09
30	2,000	0.2	5	00:11	5	00:12	30	35	2,000	4	00:09	3	00:10
50	3,000	0.3	5	00:13	5	00:16	50	52	3,000	5	00:13	5	00:14
150	10,000	1.0	7	00:51	8	01:05	150	169	10,000	5	00:27	7	00:48
300	30,000	3.0	17	04:02	25	05:49	300	502	30,000	11	02:19	34	04:20
500	50,000	5.0	41	11:19	59	15:47	500	835	50,000	20	05:04	44	10:20
1200	100,000	10.0	303	77:22	271	77:45	1200	1,669	100,000	68	20:51	138	39:21

Table 2 depicts the time observations obtained during the election and key generation process. The top part of that table shows the lower elected manager

number range of the executions while the lower part shows the higher elected manager number (notice the number of managers are higher in the lower part of the table). This was used to derive the average time per machine in the optimized results.

Optimized Results

Once the VHDL results were obtained, individual performance statistics using average time and average number of managers for each network size were used to build the models for shortest overall epoch completion time with least managers elected, those results are shown in table 3.

Table 3. *Optimized Multiple Manager Model*

	Total	MM's	**Election**		**Generation**		**Notificatio**	**Epoch**
OBS	Nodes	Blocks	Mgrs.	Time	#Keys	Time	Time	Time
	Key Size: 256 bits							
1	1,000	0.1	59	00:35	19	00:35	01:08	02:18
2	2,000	0.2	91	00:36	35	00:36	01:28	02:40
3	3,000	0.3	108	00:35	52	00:35	01:51	03:01
4	10,000	1.0	234	01:28	169	00:52	02:51	05:11
5	30,000	3.0	213	02:52	502	02:55	09:23	15:10
6	50,000	5.0	261	06:04	835	02:55	12:46	21:45
7	100,000	10.0	273	15:14	1,669	06:15	24:25	45:54
	Key Size: 512 bits							
8	1,000	0.1	59	00:35	19	00:39	01:08	02:22
9	2,000	0.2	82	00:36	35	00:39	01:38	02:53
10	3,000	0.3	109	00:38	52	00:41	01:50	03:09
11	10,000	1.0	220	01:39	169	01:26	03:02	06:07
12	30,000	3.0	203	04:28	502	05:02	09:51	19:21
13	50,000	5.0	240	07:41	835	05:29	13:53	27:03
14	100,000	10.0	257	16:55	1,669	08:56	25:56	51:47

Table 3 depicts the optimized number of managers admitted in relation to the total epoch time based on average times and average machines from the raw results obtained from the vivado suite. Note the million blocks (MM's Blocks) per blockchain assuming the 100 block transactional-per-node metric in the introduction to this section. The case for 100,000 blocks is shown only for a top level benchmark and will not be in the large network case Figures below (such large results inhibit visual comparison to the other large format networks).

Results Analysis and Discussion

This section describes the factors that are inherent in the model results and provides a comparative analysis between the individual observations.

Overall Factors

The results above give some very good insight into the behavior of the manager model during epoch changes. In the first instance there would need to be an overlap between one epoch finishing and another one beginning (the time it takes to update the network). This would need to be minimized so as to contain risk of attack and confusion in the nodes on the network. The overlap in that case however creates a constraint as to the elapsed time of the epoch regeneration, the old and new epoch overlap cannot be less than the epoch change.

Another important factor is the number of managers elected. In the "unitary" case, the number of managers elected are 2 and may only be feasible in networks that are less than 1,000 nodes due to the time to effect the epoch change. In networks of greater size (above 1,000 nodes) the "gang" approach is almost required, this brings its own challenges. A first challenge is to determine the number of managers to admit (the current algorithm provides that but it is based on assumptions of number of blocks and no duplicates). Furthermore, how should the key generation be partitioned among the machines, given partitioning itself would mean constraining the randomization of keys along some boundaries and reduce the true random aspect. Finally, what happens if managers are unavailable and what is the impact on the completion time (given machines used and their availability characteristics within IoT parameters).

A significant component of the epoch change is the notification of nodes due to network communication (1 second per message). This is sequential and no concurrency has been assumed for the machines' handling of multiple sessions (based on prior work, the machines are able to execute a key exchange in 0.05 seconds), in theory the machines could manage multiple sessions (one for each incomplete key exchange/payload) however, it would result in additional memory and complexity that would increase resource usage and potentially impact a primary service objective.

Finally, the results are derived from the simulator in vivado which is dependent on several operating memory factors (other items that could be working in the MS machine environment). Some of this is reflected in results being slightly different in one run versus another. These results are directional and do not substitute for field trials with actual machines populated in a network. This was infeasible given resources available and is a constraint of this research.

Individual Observation Analysis

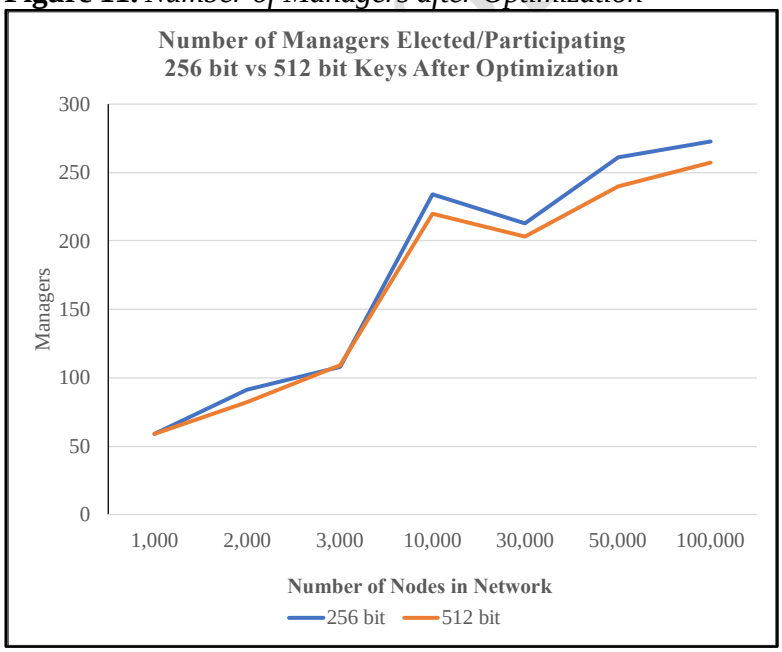
As mentioned above, the vivado simulator results are good from a directional perspective and they are useful in gauging the feasibility of the model to operate in certain IoT conditions. The unitary case is feasible mostly for networks under 1,000

nodes and may very well work if something like a network of cameras around a manufacturing plant is being implemented. On the other hand, it may be infeasible for a transactional network of individuals in large volumes of crypto currency.

In the small network sizes (1,000-3,000 nodes) the number of managers increase consistently between the three alternatives and key sizes. The number of managers elected is smaller in the case of the 512bit key size due to the additional computation that is required in order to determine the closeness of the Security Key (SK) to the random election number (since each key must be compared for and determined if it lies in the target number of managers to be elected). A similar pattern is also replicated and magnified in the large network cases. In the 100,000 node network, the election of managers tapers off significantly when compared to the 50,000 node network. This is because of computation of differences in a much larger number of memory blocks.

In the case of generation of keys the number of keys to be generated between the two key sizes is the same in both cases and the variations in completion are due to a combination of the number of keys, the number of managers generating keys and the number of blocks to traverse. In this case, each Security Key (SK) must be compared for uniqueness with previously generated SKs in the blockchain with the computation being higher for the larger 512bit key (twice as many bits to evaluate). On a more detailed basis, comparing observation 12 and 13 in 512 bit key, where the completion times for key generation are close we can see that one alternative has more machines and more keys to generate but it is not that different than the one that has less managers and less keys to generate on a smaller network footprint.

Figure 11. *Number of Managers after Optimization*



As mentioned above, the optimization model finds the smallest number of machines that will yield the lowest completion time given the raw inputs. When the optimization is on the total outcome on a multiple component model that may

penalize some of the components in order to minimize the overall time. It is for this reason that we see these differences between the the observations at different network sizes in figure 11. In that graphic, we see how the number of managers increases on a positive slope until it dips at the 30,000 level and then turns positive again after that. We can also witness how the number of managers elected stays within the range of 213 to 273 for much larger networks after that.

Figure 12. *Optimized Completion Times by Key Size and Network Scale for Small Networks*

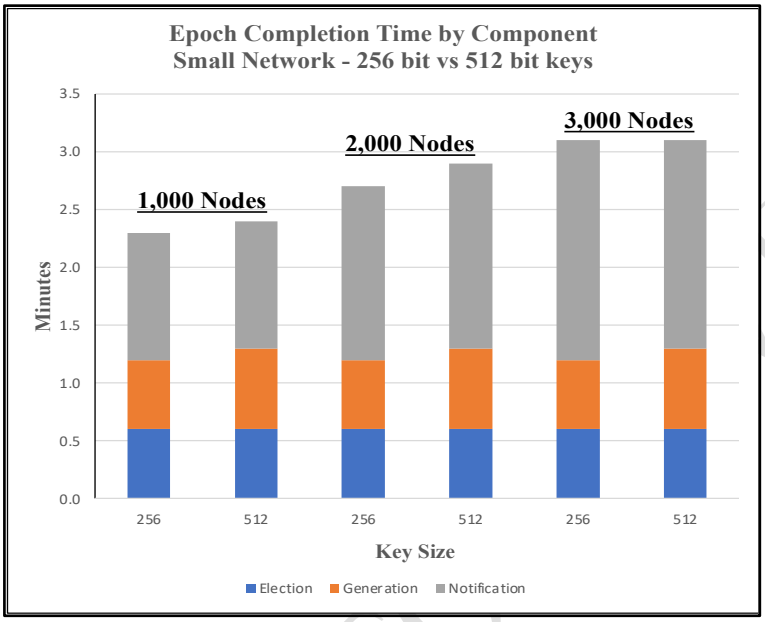
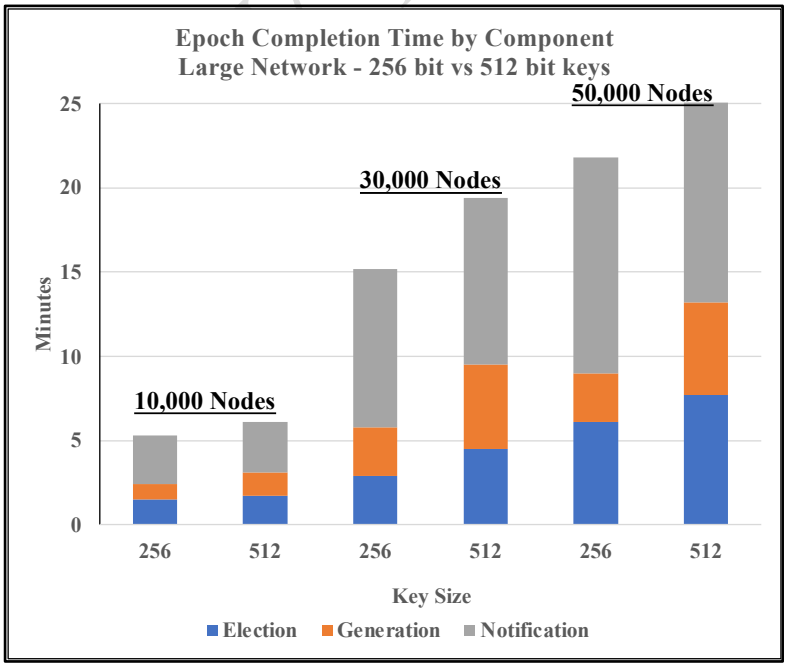


Figure 13. *Optimized Completion Times by Key Size and Network Scale for Large Networks*



When we direct our attention to the overall results we can observe how the epoch completion times increase on a fairly linear basis for each network size with some impact from key size in Figures 12 and 13 below. This relationship is what would be expected from a scaling model and may yield predictive aspects as to how the machines would behave if the blockchain networks would be built as described.

Conclusions

This study focuses on the feasibility of epoch change in small factor IoT devices configured as managers, the potential limitations on number of nodes and key lengths in a zero-trust network (presumably the internet or large extranets). It appears that the feasibility of machine operation could be potentially scaled to 50,000 nodes with value exchanges of 100 blocks per node if the overlap between one epoch beginning and the previous one ending was under 30 minutes. This requirement would need to be validated with the objectives of the protection and Transactionality required by the assets or processes using such a configuration. In reality however, most IoT networks that operate on an independent basis seem to be much smaller in size. However, given the resource requirements and potential speed of epoch change a network with 100,000 nodes could also be feasible if the machines were able to concurrently process without impacting their primary service objectives (the reduction in epoch change time would be dramatic).

It might seem that the “safe-zone” for operation could be in the 10,000 and under node range with a much smaller SLA for epoch change. If a smaller network size was selected, and cyber-attacks noted required an epoch change, it could be more efficiently managed than in the case of the larger format networks.

Most of the common attacks presented can be overcome by the key dialogues mentioned and eventual epoch change. The only attack that could prevail and enter the network would be the hijacker. In order to remediate that vulnerability, there would need to be an additional protocol for the Private Key (PK) which is under research for future studies.

References

1. Alharby, Sultan, et al. "The security trade-offs in resource constrained nodes for IoT applications." International Journal of Electronics and Communication Engineering 12.1 (2018).
2. Wazid, Mohammad, et al. "Security in 5G-enabled internet of things communication- issues, challenges, and future research roadmap." IEEE Access 9 (2020).
3. Khan, John A., and Md Minhaz Chowdhury. "Security analysis of 5g network." 2021 IEEE International Conference on Electro Information Technology (EIT). IEEE, 2021.

4. Muralidharan, S. et. al.: "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains", https://arxiv.org/pdf/1801.10228.pdf, last accessed March 19, 2019.
5. E. Knapp, J. Langill: "Industrial Network Security, Securing Critical Infrastructure Networks for Smart Grid, SCADA and other Industrial Control Systems, 2nd edition" Syngress Elsevier, Waltham, Massachusetts (2015).
6. S. Nakamoto; "Bitcoin: A Peer-to-Peer Electronic Cash System" www.bitcoin.org, last accessed March 19, 2019.
7. Medellin, J. "Exploiting Efficiencies in IoT Key Exchanges Through Reversible Logic Blockchains." 7th EAI International Conference, iCETiC 2024, Essex, UK, August 15–16, 2024.
8. Medellin, J. "Generation, Regeneration and Validation of Binary Secret Keys through Blockchain in IoT Devices" Athens Journal of Sciences Vol. 9, No.1, 2022.
9. S.Y. Huang, K.T. Cheng: "Formal Equivalence Checking and Design Debugging" Kluwer Academic Publishers, Norwell, Massachusetts (2002).
10. https://www.elprocus.com/basic-logic-gates-with-truth-tables/ accessed on 16 February, 2021.
11. R. Johnsonbaugh: "Discrete Mathematics, 8th edition" Pearson Education, Inc., New York, New York (2018).
12. Ongaro, D., Ousterhout, J. "In Search of an Understandable Consensus Algorithm" Proceedings ATC'14 USENIX Annual Technical Conference (2014), USENIX.
13. https://archlending.com/glossary/proof-of-elapsed-time accessed on 14 April, 2025.
14. N. Ferguson, B. Schneier, T. Kohno: "Cryptography Engineering, Design Principles and Practical Applications" Wiley Publishing, Inc., Indianapolis, Indiana (2010).
15. V. Pedroni: "Circuit Design with VHDL, 3rd edition" Massachusetts Institute of Technology Press, Cambridge, Massachusetts (2020).
16. R. Haskell, D. Hanna: "Digital Design Using Diligent FPGA Boards; VHDL edition" LBE Books, Auburn Hills, Michigan (2018)
17. https://digilent.com/shop/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/?srsltid=AfmBOor_jHsGMqKH-i6a0Gpsl1S9m3ClZ7EA6PnNgMSCNnrj6JgL4-Fe accessed on 14 April 2025.
18. W. Stallings: "Cryptography and Network Security, Principles and Practice 7th edition" Pearson Education Limited, London, United Kingdom (2018).

Appendix

A1. VHDL Code for Election

```
-- ------------------------------------------------
-- Header information
-- ------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
use ieee.math_real.all;    -- for uniform & trunc functions


entity ElectionProcess is
--x generic(
--x  N           : integer := 8);
  port (
      clk     : in    std_logic;
      data_io  : out    std_logic_vector (1 downto 0)
      );
end ElectionProcess;


architecture Procedural of ElectionProcess is
   -- Define a function to generate a pseudo-random number
   function rand_int(seed : integer) return integer is
      variable x : integer := seed;
   begin
      x := (x * 1103515245 + 12345) mod 214783648;
      return x;
   end function;
   signal seed : integer := 42;
  type array_1kx16 is array (15 downto 0) of std_logic_vector(511 downto
0); -- change key size here (1 of 3)
   -- declare the RAM.
   signal ram : array_1kx16;

   -- intermediate RAM output
    signal dat_out   : std_logic_vector(511 downto 0);-- change key size here
(2 of 3)
    signal w_sum    : integer :=0;
    signal end_run  : std_logic;
begin
 process
      variable num_mgr  : integer :=2; -- number of managers
      variable key_size : integer :=512; -- change key size here (3 of 3)
```

```
            variable num_mach : integer :=100000; -- number of machines in
network
            variable val_yn   : integer :=100; -- num blocks per machine/epoch
            variable adder1   : integer :=1;
            variable adder2   : integer :=1;
            variable rand_num : integer;
            variable array1   : std_logic_vector(511 downto 0);
            variable array2   : std_logic_vector(511 downto 0);
            variable array3   : std_logic_vector(511 downto 0);

          begin
        -- Instantiate RAM
        --x                    256                    ram(1)                <=
"1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111";
        --x 512
        ram(1)                                                              <=
"1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111";

          for aa in 0 to num_mgr-1 loop -- # of managers being elected
            -- Generate random integers
            for k in 0 to key_size-1 loop -- # times for key size
            rand_num := rand_int(seed) mod 100;
            seed <= rand_int(seed);
            rand_num := rand_int(seed) mod 100;
            -- Generate random real numbers between 0 and 1
            rand_num := rand_int(seed);
        --x              report "Random real: " & real'image(real(rand_num) /
real(217483648));
            -- Generate random real numbers between 5 and 10
            seed <= rand_int(seed);
            rand_num := rand_int(seed);
        --x        report "Random real (5-10): " & real'image((real(rand_num) /
real(214743648)) * 5.0 + 5.0);
            array1(k) := '1';
            array2(k) := '0';
            end loop; -- end loop key size
```

```
 for n in 0 to num_mach-1 loop
  for j in 0 to val_yn-1 loop

     -- reads
     dat_out <= ram(1);

    end loop;
-- xor mem array and array1 into array3
array3 := array1 XOR array2; -- xor two arrays into a third
 for m in 0 to key_size-1 loop -- key size (number of bits to add up)
  if(rising_edge(clk)) then
  w_sum <= adder1 + adder2;

  -- register output

  end if;
  end loop;
  end loop; -- end  machines in blockchain
  end loop; -- end # of managers elected
     data_io <= "01";
   wait ;
   end process;
end Procedural;
```


A2. VHDL Code for Key Generation

```
-- ---------------------------------------------
-- Header information
-- ---------------------------------------------
```

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.all;
4   use IEEE.NUMERIC_STD.all;
5   use ieee.math_real.all;    -- for uniform & trunc functions
6
7
8   entity KeyGen is
9    port (
10       clk     : in    std_logic;
11       data_io  : out    std_logic_vector (1 downto 0)
12       );
13  end KeyGen;
14
15
16  architecture Procedural of KeyGen is
17     -- Define a function to generate a pseudo-random number
18     function rand_int(seed : integer) return integer is
19        variable x : integer := seed;
20     begin
21        x := (x * 1103515245 + 12345) mod 214783648;
22        return x;
23     end function;
24     signal seed : integer := 42;
25
26  --x  Declare memory
27   type array_1kx16 is array (15 downto 0) of std_logic_vector(255
28  downto 0); -- change key size here (1 of 3)
29    -- declare the RAM.
30    signal ram : array_1kx16;
31
32     signal dat_out    : std_logic_vector(255 downto 0);-- change key
33  size here (2 of 3)
34     signal w_sum    : integer :=0;
35  begin
36   process
37       variable num_nodes   : integer :=1; -- number of nodes
38  admitted
39       variable num_mgr    : integer :=2; -- number of managers
```

```
        variable key_size : integer :=256; -- change key size here (3 of
3)
        variable num_mach : integer :=100000; -- number of machines
in network
        variable val_yn   : integer :=100;
        variable num_syn : integer := 1; -- number of synonims found
in the network
        variable adder1   : integer :=0;
        variable adder2   : integer :=1;
        variable rand_num : integer;
        variable array1   : std_logic_vector(255 downto 0);
        variable array2   : std_logic_vector(255 downto 0);
        variable array3   : std_logic_vector(255 downto 0);
     begin
-- populate ram
--x                512                ram(1)                <=
"1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
111111111";
--x 256
ram(1)                                                            <=
"1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111";

 -- FOR NUMBER OF NODES BEING ADMITTED
    for aa in 0 to (num_nodes/num_mgr)-1 loop -- # of nodes being
admitted divided by number of managers

 -- FIRST GENERATE THE CK AND THE PK

```

```
        -- Generate random integers
        for k in 0 to key_size-1 loop -- # times for key size
        rand_num := rand_int(seed) mod 100;

        seed <= rand_int(seed);
        rand_num := rand_int(seed) mod 100;
        -- Generate random real numbers between 0 and 1
        rand_num := rand_int(seed);
        -- Generate random real numbers between 5 and 10
        seed <= rand_int(seed);
        rand_num := rand_int(seed);
        array1(k) := '1';
        array2(k) := '0';

        end loop; -- end loop key size

-- RUN THIS LOOP BASED ON NUMBER ON SK SYNONIMS
FOUND IN NETWORK

   for l in 0 to num_mach/10000 loop -- set to find .0001 synonims

 -- GENERATE A NEW SK
        -- Generate random integers
        for k in 0 to key_size-1 loop -- # times for key size
        rand_num := rand_int(seed) mod 100;

        seed <= rand_int(seed);
        rand_num := rand_int(seed) mod 100;
        -- Generate random real numbers between 0 and 1
        rand_num := rand_int(seed);
        -- Generate random real numbers between 5 and 10
        seed <= rand_int(seed);
        rand_num := rand_int(seed);
        array3(k) := '1';

        end loop; -- end loop key size

   -- execute read/writes
 for n in 0 to (num_mach*val_yn)-1 loop -- blocks per machine and
number of machines in network
```

```
1
2   -- perform reads
3         dat_out <= ram(1);
4
5   array3 := array1 XOR array2; -- xor two arrays into a third
6    for m in 0 to key_size-1 loop -- key size (number of bits to add up)
7    if(rising_edge(clk)) then
8
9    w_sum <= adder1 + adder2;
10
11   end if;
12   end loop; -- add up differencesperform reads
13   end loop; -- number of machines * blocks per machine
14   end loop; -- number of synonims
15   end loop; -- end # of nodes needed
16        data_io <= "01";
17     wait;
18     end process;
19  end Procedural;
20
21
22
23
```