Don’t Forget to Lock your SIB: Hiding Instruments using P1687

Jennifer Dworak⁺, Al Crouch⁺, John Potter⁺, Adam Zygmuntowicz⁺, and Micah Thornton⁺
⁺Southern Methodist University, Dallas, Texas, USA
+++Asset InterTech, Inc., Richardson, Texas, USA

Abstract
IEEE P1687 is a valuable tool for accessing on-chip instruments during test, diagnosis, debug, and board configuration. However, most of these instruments should not be available to an end user in the field. We propose a method for hiding instruments in a P1687 network that utilizes a “locking” segment insertion bit (LSIB) that can only be opened when pre-defined values, corresponding to a key, are present in particular bits in the chain. We also introduce “trap” bits, which can further reduce the effectiveness of brute force attacks by permanently locking an LSIB when an incorrect value is written to the trap’s update register. Only a global reset will allow the LSIB to become operable again. In this paper, we investigate the cost and effectiveness of LSIBs and traps in several different configurations and show that these relatively small modifications to the P1687 network can make undocumented instrument access exceedingly difficult.

1. Introduction
Many of today’s integrated circuits contain embedded instruments to aid in the test, debug, and configuration of both the IC and the board on which it is placed. An IC may contain built-in-self-test (BIST) engines, trace buffers, hardware monitors, assertion checkers, error detection circuits, and event counters. On-chip sensors may monitor temperature, voltage, or delay. Configuration bits may be used to tune external interfaces (like SerDes) or internal voltage or clock frequencies. Depending on the application, data may be stored on-chip, including chip ID’s, codes, and encryption keys. It is often important that unauthorized users do not access these instruments or data. An effective low cost method is needed to provide access only to authorized users and to make these functions inaccessible once the board is in its final-use environment.
P1687 [1] is a proposed IEEE standard that provides effective and efficient access to on-chip instruments. An extension of the boundary scan standard IEEE 1149.1 [2], P1687 provides multiple benefits. Among these is that it allows a standard IEEE 1149.1 TAP (Test Access Port) controller to access a variety of instruments in different configurations without needing new instructions to encode each configuration. Instead, the scan chain may be dynamically reconfigured for instrument access by using data within the chain itself. Specifically, a special scan cell, a Segment Insertion Bit (SIB), may be added to the scan path. When the enabling logic state is clocked into the update register of the SIB, the SIB opens a new chain segment, providing access to new instruments or data.

In unsecured P1687, anyone who can shift appropriate data into the SIB and cause the 1149.1 TAP controller state machine to cycle through the UpdateDR (Update Data Register) state is capable of opening any segment of the chain and accessing any of the instruments enclosed within. If an attacker suspected that the chip contained a P1687 network and wanted to investigate the internals of the chip or find particular information or instruments behind SIBs, he could simply ensure that during his investigation each scan cell in the chain was updated with a logic 1 (or 0) at some point. If one or more of the scan cells was actually a SIB, the scan chain length would change, and this change could be observed by an attacker shifting known sequences of bits through the chain.

To make such an investigation more difficult, we propose a new type of SIB, a locking SIB (LSIB), that can only be opened when particular bits in the scan network are set to pre-defined values that form an “embedded key.” We also introduce a trap bit that can generate a SIB-disabling signal. Once triggered, this disabling signal will prevent a locked SIB from being opened (or closed) even if the correct key is selected. Only a global reset can reset the trap and allow the SIB to be opened.

In this paper, we show how SIBs may be modified to serve as locks in a P1687 network. We consider ways that an attacker may use to explore the network and attempt to open locked SIBs for unauthorized instrument access. We probabilistically estimate the chances of success and the time required to open a locked SIB for different network configurations. We show that for relatively little overhead, we can significantly increase the difficulty an attacker faces. To the best of our knowledge, this is the first paper to propose modifying SIBs that are present in a P1687 network to hide the very existence of those SIBs when appropriate data is not present in the scan network.

2. Previous Work
IEEE 1149.1 [2] was originally proposed to allow effective testing of the interconnections between chips on a board. It describes a TAP (test access port) that defines four pins

---

¹ This paper was supported in part by NSF grants CCF-1110290 and CCF-1061164.
used by a tester to interact with the scan network: TDI (test data in), TDO (test data out), TMS (test mode select), and TCK (test clock). It also defines an 1149.1 state machine, which allows instructions to be shifted into the instruction register and data to be shifted into the TDR (test data register). State machine transitions are controlled by the value of TMS on the rising edge of TCK. The test data register (TDR) corresponds to the scan cells on the chain, and these historically correspond to scan cells on the input and output pins of a chip on a board. IEEE 1149.1 defines several instructions as mandatory, including BYPASS, EXTEST, and SAMPLE. RUNBIST and INTTEST are optional, and designers often add custom instructions to the controller to aid in configuration, test, or debug.

Many JTAG attacks look for “private” instructions in an 1149.1 implementation. This is fairly easy because the JTAG instruction register has a stated size, and the JTAG documentation (the BSDL file) lists “public” instruction encodings and the size of the instruction register. An attacker only needs to shift in undocumented encodings and pass through UpdateIR (Update Instruction Register) in the 1149.1 state machine to activate undocumented instructions. If the instruction is unused, it generally maps to a BYPASS instruction (where a single shift-capture bit is placed in the TDI-TDO scan path). However, in many cases, undocumented instructions are actually hidden test and debug features, e.g. BIST modes, the manufacturing Scan and Scan Compression modes, or debug modes or functions. Altera actually lists a few private instructions in its manuals, with no explanation other than a warning that using these instructions could destroy the chip [3].

Sometimes, security-related data/functions are also hidden behind undocumented instruction encodings. To prevent hacking, a common response is to disable the JTAG port by fusing-off the TMS signal (permanently placing the JTAG state-machine in the Test-Logic-Reset state) after IC test completes. All JTAG functions are disabled—even public functions for board test and software development debug functions become unavailable. Because the JTAG port is often necessary for board/system test, debug, and integration, this is unacceptable.

To protect a chip from unauthorized JTAG attacks, several researchers have suggested restricting access using challenge-response pairs (e.g. [4], [5], [6]). For example, the author of [5] proposed adding hardware for generating challenge-response pairs and computing hashes to prevent unauthorized JTAG or instrument instructions from being executed. Several ring oscillators are used to generate a random challenge, and a SHA256 hash engine is used to compute the hash with an internally known secret key. A user must correctly compute the same hash value externally to process the instruction. In [6] a 4-level security protocol uses challenge-response pairs and encryption to provide varying levels of security. In [7], a security module and test control module are used to protect memory content. If the security module is in restricted mode, then only limited access to the memory is allowed, data encryption and decryption of the memory contents is disabled, and the TDO pin is forced to a logic 1.

In [8], [9] challenge-response pairs are part of a system that contains a Security Authentication Module (SAM) and an “Access Monitor (AM). On boot-up, the SAM locks the JTAG interface, only allowing access to the SAM instruction register. An authentication protocol is then used to unlock the interface and assess privilege level. The AM restricts access to portions of the scan-chain by blocking UpdateDR (Update Data Register) if the user privilege level is lower than that stored in a memory.

There has also been considerable interest in ways that internal scan chains can be used to break encryption hardware (e.g. [10], [11], [12]). To help avoid this, the authors of [13], [14], [15] divide the scan chain into segments and reorder the segments of the chain if the first $k$ bits shifted in do not correspond to a pre-chosen “key,” making it more difficult to control and observe. The authors of [16] add a reset operation to prevent an attacker from switching from normal mode to test mode without resetting the data in the scan flops. In [17], the flip-flops only capture data if certain bits of the test vectors (hopefully don’t care bits) are set to pre-defined values. In [18], some bits in the chain are dynamically inverted depending on previous output values of selected flip-flops, while [19] inserts random inverters before the shift input on some flops to make extraction of information from cryptographic hardware more difficult. In [20] scan data is also selectively changed, but with XORs. Note that some of these methods don’t necessarily prevent access to data; they simply make it more difficult to decode and use.

The authors of [21] aim to protect a scan chain by requiring several keys to be scanned into the chain over several cycles during a test initialization phase before testing begins. Finally, [22] proposes an open circuit deadlock (OCD) cell which inserts an open into the scan chain if a SecureRST signal has not been asserted by key checker function hardware.

Our proposed methodology complements previous approaches while harnessing the capabilities of a P1687 network to add new layers of security. Locking SIBs with keys makes discovering even the existence of an undocumented SIB, and thus whatever lies behind that SIB, more difficult. Including traps actively works against brute force attacks and scan chain investigation by making it impossible to ever access parts of the architecture if an incorrect value is present in a trap during an update operation. The attacker will not know that the trap has been sprung, and will have no way of clearing it unless the chip’s entire JTAG/IJTAG architecture is reset. Our approach also allows multiple layers of locks, traps, and keys of arbitrary complexity to be inserted to provide increasing levels of security. In addition, it provides opportunities for the insertion of “honeypots” that make an
attacker incorrectly think he is making progress. Finally, our method does not require expensive encryption schemes or schemes that shut off all test hardware, but relies on data naturally shifted through the chain to allow fine-grained access to particular instruments.

3. Accessing instruments with P1687

IEEE P1687 is intended to provide an efficient method for accessing embedded instruments as their numbers increase and “instruction-based access” under 1149.1 becomes unwieldy. Another goal is to define the documentation so that it better describes the embedded instruments using the Instrument Connectivity Language (ICL) to describe the serial scan path access and the Procedure Description Language (PDL) to describe the instrument operations, making instruments portable in use. This allows automated retargeting of embedded instrument operations (PDL) from the instrument interface to the chip pin boundary.

Although P1687 is a broad standard that allows both IEEE 1149.1 and IEEE 1500 hardware structures to be supported, when used with ICL and PDL, there is a recommended architecture. It is based on a variable length scan path accessed through a single 1149.1 instruction. For example, consider Figure 1. Here, the box on the left is the IEEE 1149.1 TAP (test access port) controller, including a Bypass register, instruction register (IR), and the 1149.1 finite state machine (FSM). A P1687 register in the TAP provides access to the P1687 network and the embedded instruments within. The P1687 network can be dynamically reconfigured by opening and closing Segment Insertion Bits (SIBs). Figure 2 shows a SIB schematic.

When the SIB in Figure 2 is closed, the value in the Update cell and thus the value of both Select* signals is equal to 0. (The Select* signals are connected.) When closed, data passes from the TDI1 input to the shift cell, and from the shift cell to TDO1 on the rising-edge of TCK when the 1149.1 state machine is in the ShiftDR state (ShDr=1). If all the SIBs in Figure 1 were closed, the P1687 scan network would appear to contain only three scan cells, the three SIBs on the far left of the hierarchy.

To allow access to a new area of the network, including the embedded instruments of Figure 1, one or more SIBs must be opened. For example, accessing the top instrument of Figure 1 requires opening 3 SIBs. Opening the SIB in Figure 2 occurs when a value of 1 is clocked into the Update Register on the falling edge of TCK when the 1149.1 state machine is in the UpdateDR state (UpDr=1). This allows Select* (which also serves as an enable for cells in the new chain segment between TDI2 and TDO2) to be set to 1. The value from TDO2 will be clocked into the shift cell when the 1149.1 state machine is in the ShiftDR state, and the data at the TDI input will be passed to the new chain segment through TDI2. Once a new value is put in the Update cell, that value remains until the state machine again executes an UpdateDR or until Global Reset is pulled. Also note that the SIB in

Figure 2 is only one example of a SIB. Other variations (such as opening the SIB when the Update Cell is equal to 0 or inserting the new chain after the SIB) are possible.

4. Example Attack Scenarios

Attackers of varying sophistication could breach the security of instruments in a P1687 network. For example, as part of a scheme to help detect counterfeiting, a company may place chip ID information behind a SIB. This ID may later need to be read by an authorized company representative investigating a non-working board that is suspected of being counterfeit so that the ID information can be checked against a secure database. Obviously, it is desirable to hide this ID from counterfeiters. If the ID was not created with a physically unclonable function, an attacker could read the ID from a chip and then clone that ID and the ID access mechanism in their counterfeit devices. It may not even be necessary for the attacker to know that this is the ID, provided that he can make his counterfeit chip behave like the original.
As a result, an attacker who knows or suspects that an ID access method exists may send bits through the scan chain while ensuring that each scan chain element experiences an UpdateDR while a logic 1 is present in that element. For example, the attacker may shift the pattern 10000… through the chain before performing an UpdateDR. If the chain becomes longer, the first bit, which contained a 1, may be a SIB that opened a new portion of the chain. Next, the attacker may send the pattern 010000… through the chain. A methodical attacker can explore the entire P1687 logic network, map the network, and observe the circuit behavior initiated by an UpdateDR as well as data captured in the network on a CaptureDR.

In addition to chip IDs there are also many other chip components that may need to remain inaccessible. Trace buffer or scan chain data may be used to obtain state information for breaking software encryption. Access to on-chip configuration bits opens a chip to sabotage or unlicensed usage. Accessing and running all of the BIST engines at once could potentially destroy the chip.

We aim to make attacks of this nature harder and to discourage/prevent attacks by low and medium grade attackers. By themselves, the proposed methods are not a silver bullet solution for protecting something absolutely critical, (such as codes that set off a nuclear weapon), but they can help make the cost of accessing and mapping private areas of the P1687 network much more expensive.

5. Locks, Keys, and Traps

5.1 Locks

If a SIB is made so that the UpdateEn signal cannot be delivered until specific logical conditions on other signals are met, then the SIB can be “locked” so it cannot be opened. We call a SIB that can deny the “update function” a locking-SIB, or LSIB. An LSIB may be placed anywhere a SIB may be placed. It can be made in place by placing, for example, an AND-gate across the UpdateEn signal (shown as UpDR in Figure 3).

![Figure 3: Addition of AND gate to a SIB creates an LSIB. (Note that some key bit inputs may be inverted.)](image)

The key bits (KBITS) shown in Figure 3 are derived from the outputs of bits in the accessible scan path. They could be part of the original chain or could be added to the chain for this purpose. Note that although the figure shows only a single AND gate being used to control passage of the UpdateEn signal that allows the LSIB to open on an UpdateDR event, other logic equations are possible. For example, some key bits may need to be set to zero while others are set to one to open the LSIB.

![Figure 4: Public and Secure P1687 Instrument Access](image)

Figure 4 shows a representation of a P1687 N-Bit long test data register with two SIBs that provide access to scan path registers that are used as Instrument Interface Registers (I-IF TDR). In Figure 4B, one of the SIBs has been exchanged with an LSIB. The outputs of several scan cells are now routed (and possibly inverted) to provide Key-Bits to the LSIB.

5.1.1 What is the probability of opening a locked SIB with a random pattern?

Assume that the key consists of \( k \) bits in the current scan path. Those scan cells must be set to particular values to enable the opening of the SIB to allow access to the secret instrument. The probability of successfully opening the SIB with a random scan pattern is:

\[
\text{prob(opening SIB with key of } k \text{ bits}) = \frac{1}{2^{k+1}} \tag{1}
\]

where \( k \) is the number of bits in the key. The “plus one” arises from the fact that the correct bit must be present in the SIB itself when an update is performed. Note that there is no reason why an attacker must choose random vectors to enter into the chain. However, with no other knowledge of the scan network, this is one of the most effective choices. Applying biased values runs the risk of biasing guesses against opening the SIB instead of making opening more likely. Note that the attacker will not know how many bits are in the key and thus the probability of opening the LSIB. He will only know the length of the chain (which can be found using the procedure described in Section 5.1.2). However, if he applies random vectors to the chain, the non-key and non-SIB bits become don’t cares, and don’t decrease his probability of opening the lock on any given attempt.

5.1.2 What is the expected amount of time that it will take on average to open a locked SIB by an attacker who does not know the key?

Assume that the scan path contains \( n \) bits when the LSIB is not open. Thus, \( k \) of the \( n \) bits corresponds to the key and
1 of the n bits corresponds to the SIB that the attacker is trying to open. Under these conditions, the attacker must decide how often to perform an update and how to determine if the SIB has been opened. One way for an attacker to determine if a SIB has been opened is to look for an increase in the scan chain length. Thus, the first thing the attacker should do is find the initial chain length.

**Determining the initial length of the chain**

Assuming that the attacker starts from a global reset, the scan chain will initially contain whatever unknown values are present on reset. Once the attacker enters shift mode, he can begin shifting a distinctive pattern of d bits into the chain, such as 101001000…, as shown in Figure 5. Here, d is an attacker-defined number that is large enough to give him some confidence that he is really seeing the bits he shifted in eventually coming back out. After n cycles, the shifted in pattern will begin to emerge from the chain. Also, after the attacker finishes shifting in the d bits, he can shift in n random bits (designated in the figure as R’s) to entirely fill the chain with random values while still verifying that the patterns he shifted in are shifted out. The process requires n + d shift cycles and ends with a random vector in the chain.

![Figure 5: Determining the initial chain length](image)

**Figure 5: Determining the initial chain length**

**Trying to unlock an LSIB with a random vector**

Once a random vector is present in the chain, an UpdateDR must be performed to try to unlock the SIB. The chain must then be checked for a change in length. Because the attacker needs to use the 1149.1 TAP controller and state machine to accomplish this, he must go through the procedure shown in Figure 6.

Because there is already a random vector in the chain, the flow starts with an UpdateDR. If the correct values are in the keys and the LSIB register, then an UpdateDR will allow the LSIB to open, changing the length of the chain. However, to determine if the length of the chain has changed, it is necessary to run through the other cycles of the state machine until we reach ShiftDR. Note that it was necessary to go through CaptureDR, so the bits in the scan chain are no longer the random bits (R) that opened the LSIB, but the captured bits (C), as shown in Figure 7.

![Figure 7: Contents of the scan chain on a failed attempt to open an LSIB.](image)

**Figure 7: Contents of the scan chain on a failed attempt to open an LSIB.**

Once the attacker goes through the capture stage, he is ready to shift out the data and determine if the length of the scan path has changed. However, because the bits in the chain are no longer the bits he entered, he must use an identifying sequence to mark the end of the chain. Thus, the first bits shifted in consist of a set of d bits that can serve as a marker. Once the distinctive d bits begin shifting out, the attacker will know how long the chain is.

If a new segment of the chain has been opened, the d bit sequence will arrive late. If a segment of the chain was closed, the sequence will arrive early. In either case, random bits are shifted in following the d-bit sequence. Assuming that the chain did not change in length, after n+d shifts, a new random vector is present in the chain. The attacker can then try to use the new random vector to open the LSIB and returns to the top of Figure 6 as an UpdateDR is executed. Thus, once the attacker starts making guesses, the cost of a guess in cycles corresponds to the number of cycles in Figure 6:

\[ \text{Cost}(\text{LSIB unlock attempt}) = 5 + n + d \]  (2)

The expected number of random guesses required to unlock the LSIB is the inverse of Equation 1 [23].

\[ \text{Expected attempts} = 2^{k+1} \]  (3)

Thus, the expected number of cycles on average required to open a simple LSIB with random vector guesses (not counting initial setup and finding of the chain length) is:

\[ \text{Expected Cost(unlocking LSIB)} = (5 + n + d)2^{k+1} \]  (4)

**Table 1: Expected time (t) to unlock an LSIB in days**

<table>
<thead>
<tr>
<th>k</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>48</th>
<th>64</th>
<th>80</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>640</td>
<td>1280</td>
<td>2560</td>
<td>5120</td>
<td>10240</td>
<td>20480</td>
<td>40960</td>
</tr>
<tr>
<td>t</td>
<td>3.97E-7</td>
<td>1.99E-4</td>
<td>2.57E1</td>
<td>3.36E6</td>
<td>4.39E11</td>
<td>5.74E16</td>
<td>7.52E21</td>
</tr>
</tbody>
</table>

Table 1 shows the expected cost of unlocking the SIB for several values of k and n for a clock speed of 10MHz. For very small key values and chain lengths, the LSIB can be unlocked very quickly. For example, for k=16, n=1280,
and d=25, the expected time to unlock the SIB is approximately 17 seconds. A key size of 32 with n=2560 takes 25.7 days, on average, to break. However, a key size of only 48 bits with a chain length of 5120 takes over 9000 years on average.

5.2 Adding Traps

As shown in Table 1 above, the average time required to unlock an LSIB can be prohibitively expensive provided that the size of the key and the scan chain are large enough. However, it is possible for an attacker to get “lucky” and guess the key in very few attempts. To help prevent this, and to increase the cost of an unlock attempt, we propose a new scan chain element: the trap bit (TBIT).

5.2.1 What does a trap bit look like?

A trap bit is TDR-like construction that generates a signal used to disable LSIBs or KBITs or even the ability of the scan path to operate. It will self-lock and maintain that signal until a global reset is applied.

![Figure 8: Trap-Bit Schematic Example](image)

**Figure 8** shows an example Trap-Bit made from a shift-update type of scan path element that passes data from TDI to TDO on the rising-edge of TCK when the JTAG state machine is leaving the ShiftDR state. On the falling-edge of TCK, when the JTAG state-machine is in the UpdateDR state, the data from the shift-element is transferred into the update-element, thus setting the state of the TrapEnable signal (shown as TrapEn at the output of Figure 8). If the data is a logic 1, then the trap is asserted and the TrapEnable signal is inverted and used in conjunction with the AND-gate to block the trap-bit’s UpdateEn (shown as UpDR in Figure 8) so that the TrapEnable signal can only be cleared (set to logic 0) by conducting a Global Reset. The TrapEnable signal can be used for many different purposes to provide slowdown strategies to the investigator. A Trap-Bit may be used to:

- block the ShiftEnable signal to one or more bits in the chain to stop the shift operation from working.

Trap bits are very powerful and may be passive so that there is no outward indication that a trap has been sprung. In other cases, the trap spoofs behavior that may have occurred by the act of conducting an update with random data (such as closing a SIB). In the most extreme case, the trap generates a very visible action, such as blocking the scan path, disabling TCK, or factoring elements into the scan path that will cause the shift-rate to slow down.

5.2.2 How does the inclusion of traps affect the attacker’s strategy and cost?

The possible inclusion of traps in the scan chain complicates the attacker’s strategy and increases the time required to perform each guess. Specifically, because each incorrect attempt could have tripped a trap, all future UpdateDR’s may be useless, even if the attacker is lucky and finds the key. The only way to ensure that a failed vector failed because it did not contain the key as opposed to because a trap had been tripped is to pull a global reset between attempts. This means that it is no longer possible to load a new random vector while checking the length of the chain, and additional cycles through the TAP controller are needed to make each attempt. Also, a Global Reset closes all open SIBs.

Consider Figure 9. The beginning of this sequence is identical to that in Figure 6. The attacker will try to open the LSIB with an update, and then scan out the captured pattern followed by a special d-bit sequence. This d-bit sequence can be used to mark the end of the chain and determine the chain length. However, it is possible that the update also tripped a trap. Thus, if the attacker believes that traps may be present, he must now pull a global reset to clear any traps to ensure that future guesses have some chance of opening the LSIB. Thus, several more clock cycles following reset are needed to fill the chain with the next random vector. (Note that the shown sequence is only possible if the JTAG TRST* signal is supported to conduct an asynchronous reset and does not remove the P1687 instruction from the instruction register. Otherwise, the sequence must pass from Shift to Exit1 to UpdateDR to SelectDR to SelectIR to Test-Logic-Reset (TLR), then to conduct an IR-Scan to re-install the 1687 AccessLink Instruction, and back to RTI to get back into the normal DR-Scan sequence—significantly increasing the cost.)

The contents of the chain through this process are illustrated in Figure 10. The initial vector to be tried is present in the chain at the time of UpdateDR when the attacker tries to open the LSIB. After capture, the bits change to the captured data, and this data is shifted out, followed by the d-bit sequence. After reset, the contents of the register are again replaced (shown here as all 0’s), and as the attacker traverses the state machine, a new set of values are captured in the flip-flops (shown as C1).
Finally, the new random pattern is shifted into the registers in preparation for the next update.

Thus, once the attacker starts making guesses, the cost in cycles of a random guess to open the LSIB with a possible trap is:

\[ \text{Cost(LSIB unlock attempt w/\text{TRAP})} = 10 + 2n + d \quad (5) \]

5.3 Hierarchical Locks

As already noted, although it may take a prohibitively long time for an attacker to guess a key on average, it is still possible for him to “get lucky” and guess it relatively quickly. To make the network even more secure, we can use a hierarchical network structure in which KBITS, LSIBs, Trap bits, and instruments may be located on multiple levels of a scan hierarchy, and more than a single LSIB must be unlocked to access an instrument. The larger the number of difficult LSIBs that must be unlocked in succession, the “luckier” an attacker must be to get all of them. In addition, the presence of traps in a hierarchical structure increases the amount of work that an attacker must do, not only on a failed attempt, but even on a successful attempt that actually unlocks an LSIB. For an example of a hierarchical P1687 network containing locks, keys, and traps, consider Figure 11.

In Figure 11, the P1687 scan network consists of 3 Test Data Registers (TDR’s), each containing N, P, and R bits respectively. To access the 2<sup>nd</sup> TDR, which contains P bits, LSIB #1, shown in yellow, must be unlocked and opened. Similarly, to access the 3<sup>rd</sup> TDR, containing R bits, LSIB #2, shown in blue, must be opened. There is also a 3<sup>rd</sup> LSIB, shown in green, that must be unlocked to provide access to a 4<sup>th</sup> TDR that communicates with a hidden instrument. To unlock each of the LSIBs, particular key bits must be set:

- a & b to unlock LSIB #1
- c, d, & e to unlock LSIB #2
- f, g, h, i, and j to unlock LSIB #3

Note that both the level of the LSIB in the hierarchy as well as the location of its keys determines the order in which the LSIB may be unlocked. Even though LSIB #3 is at the first level, it cannot be unlocked until LSIB #1 and LSIB #2 are unlocked to provide access to the R-bit TDR that contains some of its keys. Because traps and keys may be present on multiple levels, it also becomes important for an attacker to know which bits correspond to...
keys for opening a particular LSIB. Other bits that may have been don’t cares for opening a particular LSIB could be traps or keys for opening a subsequent LSIB. If these bits are set to the wrong value, it may be impossible to open subsequent LSIBs.

For example, consider Figure 11 again. Trap bit k is on level one. Depending on the network design, this trap bit could directly prevent any of the LSIB’s from being unlocked—even LSIB’s on a different level. It could also prevent some combination of LSIB’s from being unlocked.

First, let’s assume that this Trap bit will disable only LSIB #1. In that case, an attacker who successfully opens LSIB #1 has obviously not previously tripped the trap since global reset. The same sequence can be used to open LSIB #1 after reset on subsequent tries when the attacker tries to open LSIB #2. However, consider instead what would happen if Trap bit k instead disables LSIB #2. In this case, LSIB #1 can be successfully opened with a pattern that prevents LSIB #2 from ever opening.

To prevent this, an attacker who is attacking a hierarchical locking structure and successfully opens LSIB #1 must identify which bits were the key bits for LSIB #1 as well as which bit was LSIB #1 itself and which bits were don’t cares. To identify these bits, the attacker can reset the circuit and re-execute the successful pattern while complementing one or more of the bits. If the lock still opens, the complemented bits were don’t cares. If the lock does not open, the complemented bits corresponded to either the key or the LSIB itself. In the worst case, the attacker may choose to complement each bit one at a time and apply the modified pattern after a global reset as described in Section 5.2.2. (Utilizing the global reset helps to prevent tripping a trap when one of the bits is complemented and thus corrupting the subsequent analysis.) With this method, the time required to discover which bits are “cares” and are either KBITs or LSIBs is:

\[ Cost(\text{finding care bits}) = (10 + 2n + d)n \] (7)

This corresponds to the cost of an unlocking attempt, as shown in Equation (5) multiplied by the number of bits in the scanned-in pattern: n.

Next, in a hierarchical structure, it is required to find which bit was the LSIB itself. Once the LSIB is opened, the length of the chain changes, and the absolute position of bits in the new chain is different. Thus, the location of the LSIB must be used to identify the new segment. (Note that this will eventually have to be done by an attacker who opens even a single level SIB. It just doesn’t have to be done during the unlocking search.) The attacker may accomplish this by using a special property of an LSIB: performing an update with an opposite value (generally a logic 0) in the LSIB, will allow it to close, provided that the key bits are still correctly filled in the chain at the time of the UpdateDR. Otherwise, the LSIB will remain open. Thus, one way to identify the LSIB is to take the successfully opening pattern, with identified care bits, and repeatedly apply it while complementing each of the care bits in turn to try to close it. Note that additional bits corresponding to the length of the newly opened chain must be inserted in the pattern. If a care bit does not close the LSIB, it was a key; if it does, it was the LSIB itself.

If the attacker assumes that one of the keys could have also been a Trap or that a Trap may exist in the newly opened segment, then once again, each attempt must start with a global reset. First, the LSIB must be opened. This requires 6+n cycles (the cycles from reset to UpdateDR in Figure 9. At this point, the LSIB is open, and the chain length is longer. After 4 more cycles, he can start to shift in the new pattern. Assume that the newly opened chain consists of m bits. The pattern shifted in should consist of n+m bits with the m bits inserted within the original n bits before the location of the suspected LSIB. (This assumes that the recommended architecture was followed. If not, the attacker must try inserting the m bits before and after the LSIB on separate attempts). The value clocked into the suspect LSIB will be the complement of the one that opened it. This is followed by 5 clock cycles (UpdateDR to ShiftDR) and n+m+d cycles of shifting to determine if the scan chain has changed length and thus if the LSIB was closed. Total time for a single attempt:

\[ Cost(\text{testing care bit}) = 15 + 3n + 2m + d \] (8)

If the LSIB is the last bit he tries, and if the extra chain was indeed inserted before the LSIB as the attacker suspected, and if no trap was tripped for the LSIB with the opening pattern, then the cost of finding the LSIB is:

\[ Cost(\text{finding LSIB in care bits}) = (15 + 3n + 2m + d)(k + 1) \] (9)

At this point, the attacker may still not be done because he must now try to open any LSIB that may be present in the new part of the chain. Thus, he must 1) start with a global reset, 2) load a valid opening LSIB sequence into the chain with the care bits filled appropriately and the don’t care bits filled randomly, 3) perform an update to open the first LSIB, and 4) try a new random pattern in the m+n bits of the new chain to see if it opens the next SIB. This corresponds to the cost of an attempt at opening the next LSIB and is the same as the cost of testing a care bit, as specified in Equation (8). The difference is that we are looking for an increase in length, instead of a decrease.

The probability of successfully opening the next LSIB is still based upon the number of bits in the key in the current cycle. However, it is also based on the probability that no trap bits affecting this LSIB were tripped when the first LSIB was opened with its pattern. Thus, the expected number of patterns for opening a second level LSIB with k2 keys and r2 relevant traps is:

\[ Expected\text{attempts}(\text{opening 2nd level LSIB}) = 2^{k2+r2+1} \] (10)

Thus, the cost of opening the 2nd level LSIB after the care bits for the first LSIB have been found is equal to:
Thus, the expected time required to unlock the second LSIB once the first LSIB has been opened for the first time is equal to the sum of Equations 7, 9, and 11. To this we need to add the cost of finding the first key and opening the first LSIB, which is found in Equation 6. Opening additional LSIBs will proceed similarly, with corresponding additions in cost.

6. Experimental Results

To evaluate the strength of our method, we considered the security provided under the attack methodologies described in Section 5 as well as the hardware cost required to implement the design. We first considered a simple one-level scan structure containing a single LSIB. The number of key bits varied between 8 and 96 while the number of bits in the chain varied between 640 and 40960. (Note that this necessarily a particular relationship between the length of the chain and the number of key bits. We simply allowed the key size to increase as the chain length increased.) In all of our experiments, the value of “d” (the number of bits in the distinguishing pattern that helps determine the chain length) is 25.

Table 2: Time to unlock LSIBs with traps & 2 layers of hierarchy

<table>
<thead>
<tr>
<th>K</th>
<th>N</th>
<th>Expected time to unlock LSIB w/traps (days)</th>
<th>% increase when traps are added</th>
<th>% increase when layers = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>640</td>
<td>7.79E-07</td>
<td>96.269</td>
<td>465.00</td>
</tr>
<tr>
<td>16</td>
<td>1280</td>
<td>3.94E-04</td>
<td>98.092</td>
<td>319.76</td>
</tr>
<tr>
<td>32</td>
<td>2560</td>
<td>5.13E+01</td>
<td>99.035</td>
<td>309.45</td>
</tr>
<tr>
<td>48</td>
<td>5120</td>
<td>6.69E+06</td>
<td>99.515</td>
<td>304.25</td>
</tr>
<tr>
<td>64</td>
<td>10240</td>
<td>8.76E+04</td>
<td>99.757</td>
<td>302.13</td>
</tr>
<tr>
<td>80</td>
<td>20480</td>
<td>1.15E+07</td>
<td>99.878</td>
<td>301.07</td>
</tr>
<tr>
<td>96</td>
<td>40960</td>
<td>1.50E+22</td>
<td>99.939</td>
<td>300.53</td>
</tr>
</tbody>
</table>

The expected time in days required to find the key and open the LSIB for different key sizes and scan chain lengths was shown in Table 1 for a clock of 10 MHz. The third column of Table 2 shows the expected time in days required to unlock the LSIB when the attacker assumes that traps may be present, also at 10 MHz, for various key sizes (K) and scan chain lengths (N). For very small key and chain sizes (e.g. K=8 and N=640), the time required is less than a second. However, in the case of K=32 and N=2560, the time required 51.3 days. For K=48, the average time required is over 18,000 years.

The fourth column shows the percent increase in time when traps are included versus when they are not. Adding traps almost doubles the time required to unlock the LSIB. The impact is greater for larger scan chain lengths because the effect arises primarily from the need to do an additional scan-in operation after global reset. Note that this effect will arise even if the attacker only thinks traps may be present. He must utilize the more expensive method shown in Figures 10 and 11 to prevent the chance of having all his work wasted due to a trap being tripped. The actual existence of a trap is not important.

Finally, we considered not only unlocking LSIBs, but identifying the LSIB’s and key bits once they are unlocked. Column 5 shows the increase in time required for the 2-level hierarchical case in which two LSIBs must be opened (with traps) over the one where only one single level LSIB must be opened with traps (Column 4). In these calculations, we assumed that the second scan chain contained 128 bits, and that one trap bit was present in the first scan chain that could potentially prevent the second chain from opening. As shown in this figure, the addition of the second LSIB causes more than a 300% increase in average time over a single LSIB with traps. The increase is more pronounced for the smaller chain sizes. Specifically, the addition of the new 128-bit chain has a larger relative impact in those circuits because it is a more significant percentage of the original chain length.

We also investigated the area overhead of LSIBs. The cost is dependent upon several factors, including the number of keys and LSIBs provided for protection. For our initial investigation, we mapped a baseline write-only TDR to a Xilinx Spartan-6 XC6SLX75T-3FGG676C FPGA. We then added an LSIB that could be opened by key input signals for k=16, 32, 48, and 64. Figure 12 shows % increase in LUT slices required compared to the cost of the baseline TDR for various scan chain lengths.

Figure 12: Percent increase in slice LUT’s needed for implementing locking function over baseline TDR.

In this experiment, the area overhead corresponds to the size of the AND gate needed to monitor the chain for the values of the key signals. There is a tradeoff between the length of the chain and the overhead. For short chains, overhead varies between approximately 4 and 15%, and the proposed method should only be used when the added security is really needed. However, for longer chains, the overhead becomes vanishingly small.

We also considered the effect of increasing key sizes on an LSIB implemented in an ASIC. Synopsys Design Compiler was used to synthesize the circuits with a provided saed90nm library. The percent area increase of an LSIB with a given key size (in bits) over a standard SIB can be found in Table 3. Once again, the area increase
corresponds to the size of the added AND gate. Note that the size of the original SIB is much smaller than the area of the chain itself, so even an increase of 187% in the case of an 80-bit key is miniscule.

Table 3: Percent increase in area for an LSIB over a SIB

<table>
<thead>
<tr>
<th>Key size</th>
<th>% incr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>33.1</td>
</tr>
<tr>
<td>32</td>
<td>62.7</td>
</tr>
<tr>
<td>48</td>
<td>103.2</td>
</tr>
<tr>
<td>64</td>
<td>121.5</td>
</tr>
<tr>
<td>80</td>
<td>187.0</td>
</tr>
</tbody>
</table>

7. Conclusions and Future Work

We have introduced using Locks, Keys, and Traps to hide instruments in a P1687 network and provided a sample of basic attacks and configurations. Even with these relatively low-overhead modifications, we can greatly impact the ability of an attacker to find and access instruments. For key sizes above 48 bits, the brute force approach becomes highly intractable, especially for hierarchical implementations.

Our analysis has also exposed several additional requirements for secure P1687 design. First, the capture function of the TAP controller adds significant complexity to the exploration process—requiring a complete scan-out of the chain to determine chain length. Some IC’s build their JTAG scan path networks with a “deny capture” function to assist with scan debug—this should not be allowed for secure networks. In addition, the KBITs feeding an LSIB should not open the lock with values that the circuit gets reset to naturally or with patterns that an attacker is likely to try. For example, KBITs feeding an LSIB should generally not be logic 1’s or logic 0’s. It is also possible to make opening an LSIB a “bad thing” for the attacker. For example, a requirement for opening an easy-to-open “honeypot” LSIB could involve tripping a trap—enticing the attacker to keep tripping the trap as he repeatedly opens the SIB. Because the time involved with the attack is shift-rate dependent, another strategy is to make LSIBs that slow the shift-rate of the whole chain.

In the future we will explore further enhancing P1687 security. In addition, software may be needed to efficiently test application and instrument access in the presence of locks, keys, and traps. For example, lock, trap, and key information could be encoded in an encrypted license file made specifically for each authorized user. The tool will decrypt the file using information specific to that user and generate patterns that allow the user to select and operate instruments at a high level abstraction. We will thus also investigate ways to prevent a user with valid licensed software from snooping and using information sent to the board. Some solutions to this problem include inexpensive serial data encryption, and appropriate P1687 network architectures.

8. References


