# Attacks and Defenses for JTAG

**Kurt Rosenfeld and Ramesh Karri**
Polytechnic Institute of New York University

*Editor's note:*
JTAG is a well-known standard mechanism for in-field test. Although it provides high controllability and observability, it also poses great security challenges. This article analyzes various attacks and proposes protection schemes.
*—Mohammad Tehranipoor, University of Connecticut*

**JTAG IS THE DOMINANT** standard for in-circuit test. In use for 20 years, JTAG, like many mature standards for information systems, was conceived with a friendly environment in mind. It was designed to handle the natural enemies of digital systems: faults in design, fabrication, packaging, and PC boards. JTAG's success has been largely due to the standard's economy and flexibility, and it has been extended in various directions. JTAG testing has evolved into the de facto method for in-circuit configuration and debug.

"JTAG" refers to IEEE Std. 1149.1, *Standard Test Access Port and Boundary Scan Architecture*, which came about from recommendations of the Joint Test Action Group. The companion standard, IEEE Std. 1532, *Boundary-Scan-Based In-System Configuration of Programmable Devices*, has extended JTAG even further to support on-board programming. Although JTAG has great utility in friendly environments, in even moderately hostile environments it can lead to undesirable forms of exposure.

This article discusses attacks and defenses for JTAG-enabled systems. Our goal is to provide a practical path toward better security while maintaining strict compliance with IEEE Std. 1149.1, and without significantly affecting test economics. We do not regard JTAG as a security threat: indeed, JTAG's presence on a PCB enables stronger platform security than would typically be achievable in an equivalent system without JTAG. Our view is that security problems arise when there is a discrepancy between what people expect and what assurances a given system can provide. The JTAG concept does not preclude providing protection and assurances but is typically deployed in a minimal form that provides little or no protection. In this article, we describe several ways in which an attacker can exploit a typical JTAG deployment, we review aspects of JTAG that are relevant to attack execution, and we suggest ways to defend against threats.

## JTAG overview

The JTAG protocol defines a bidirectional communication link, intended for system management tasks, with one master and an arbitrary number of slaves. As Figure 1 shows, this link supports daisy-chaining of an arbitrary number of devices. A single master device controls the protocol state of all other devices on a chain. From a functionality standpoint, the device order doesn't matter. Shared wiring and freedom over the device order reduce the burden on the PCB designer, which translates into lower cost for deploying JTAG compared with other in-system test and management solutions.

The master can initiate interaction with any of the slaves, which never initiate interactions. A chip is JTAG enabled by the presence of two essential physical components. The first is the test access port (TAP) controller—a state machine that interprets the JTAG serial protocol. The other physical component is the boundary scan register (BSR), which is interposed between the chip's core logic and I/O modules. The BSR can be arbitrarily wide, but can be preset and read serially. JTAG has three basic modes: BYPASS, EXTEST, and INTEST.

### BYPASS mode

In BYPASS mode, the serial bitstream is copied from the *TDI* pin to the *TDO* pin, delayed by one cycle of the test clock, *TCK*.


0740-7475/10/$26.00 © 2010 IEEE Copublished by the IEEE CS and the IEEE CASS

### EXTEST mode

In EXTEST (external test) mode, the BSR is connected to the pins of the chip. The BSR specifies the values to be asserted on the chip's output pins and captures the values that are present at the chip's input pins. The data in the BSR is subsequently shifted out through the *TDO* pin. The *TDO* signal is routed back to the master, either directly or through other JTAG-enabled chips.

### INTEST mode

In INTEST (internal test) mode, the BSR is serially loaded with data bits arriving on *TDI*, and the BSR contents are applied to the terminals of the chip's internal logic. The BSR captures values that are present at the output terminals of the chip's internal logic. The JTAG standard only requires supporting the basic instructions. Implementors are free to add their own instructions, and for these user-defined instructions the implementors are free to define the semantics more or less arbitrarily. Whatever the instruction, the protocol follows the same basic pattern: Enter the SHIFT_IR state, shift the instruction into the IR, then enter the SHIFT_DR state, shift the relevant data into and/or out of the DR.

## JTAG attacks

Figure 2 shows a general model of the JTAG security landscape. This model provides a way to analyze the security risks associated with a JTAG deployment. A system can have multiple potential attackers. For example, one potential attacker is the manufacturer of one of

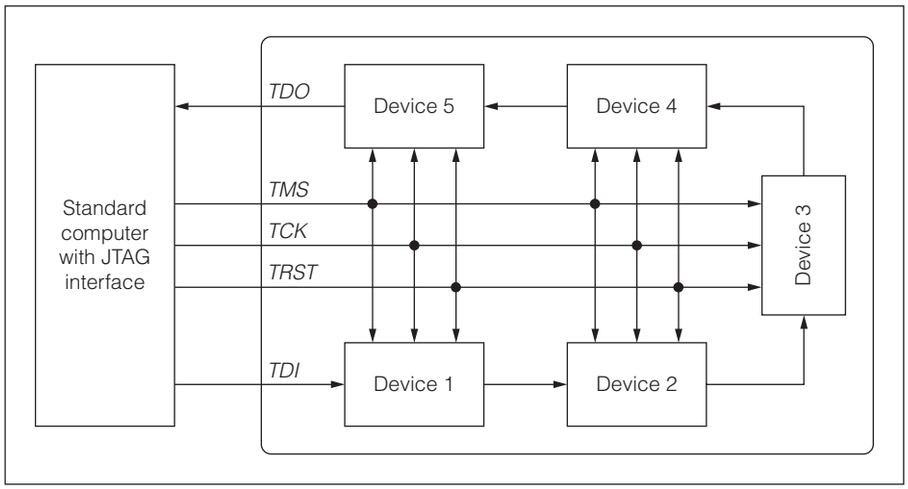


**Figure 1. The typical deployment of JTAG is a chain of several devices on a PCB. Each device can come from a different vendor. The *test_mode_select* (*TMS*), *test_clock* (*TCK*), and *test_reset* (*TRST*) signals are common to most chips. The *test_data_in* (*TDI*) and *test_data_out* (*TDO*) signals loop through the chips. The path returns to the source, which is usually either a general-purpose computer or an embedded microcontroller, functionally called a system controller.**

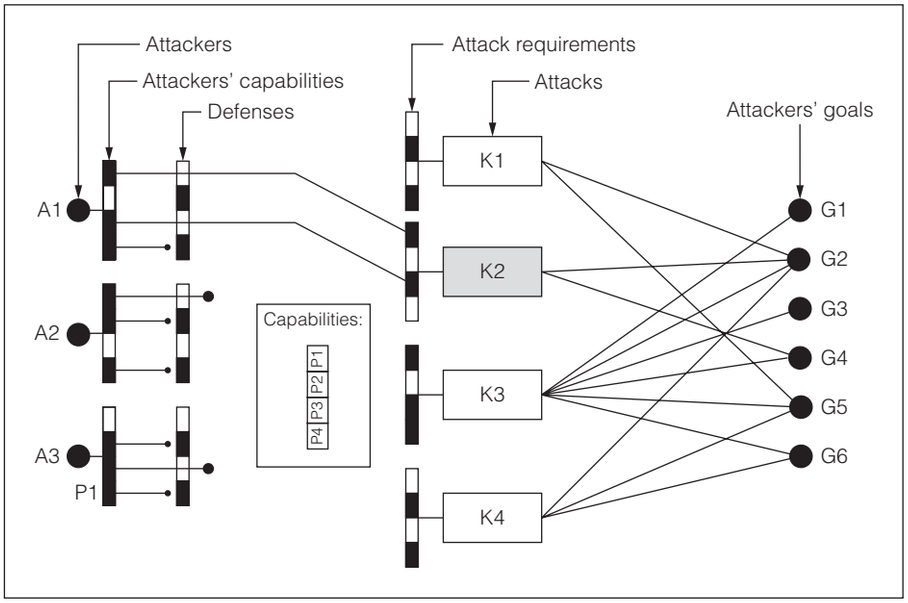


**Figure 2. Conceptual security model: A group of attackers A1, A2, and A3 have a set of goals G1 through G6. Each attacker has a set of attack capabilities, some or all of which are masked by defenses that are in place. Attacks K1 through K4 each require certain unmasked attack capabilities. Each attack can be used to reach some set of goals. This example shows that attacker A1 can achieve goals G2 and G4 because it has capabilities P2 and P4, which are the requirements for attack K2. Attackers A2 and A3 do not have sufficient unmasked capabilities to execute any of the attacks.**

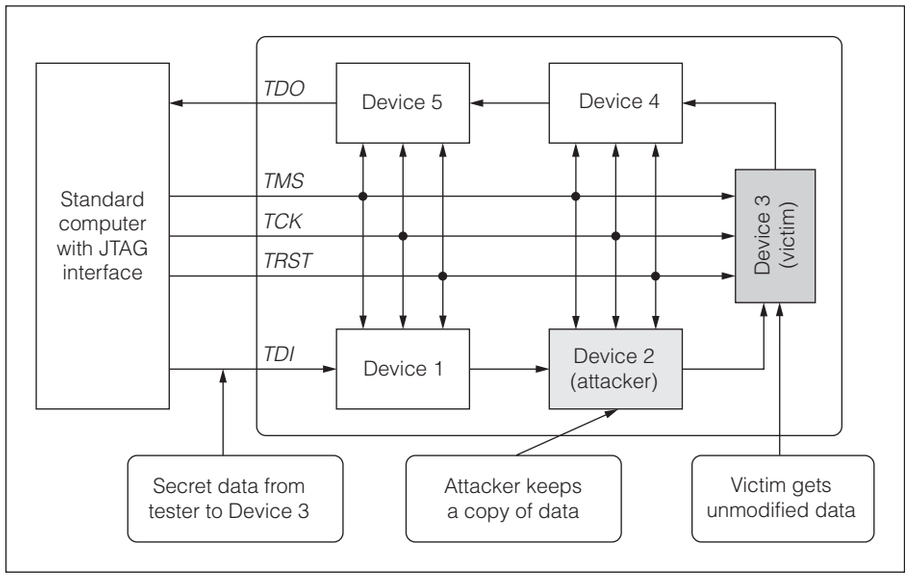


**Figure 3. The attacker obtains secret data by sniffing the JTAG data path.**

the chips. Another is a hostile end user. Each attacker has a set of capabilities, some subset of P1 through P4—one example is an attacker's ability to sniff the data sent through the data lines. Some of the attackers' capabilities might be blocked by defenses. For example, the attacker might have access to sniff the bits on the JTAG bus, but if those bits are encrypted, sniffing attacks won't work.

There is a set of possible attacks, each of which has attack requirements. For example, an attack to capture the configuration stream of an FPGA as the configuration bits are being sent over JTAG requires the ability to sniff the JTAG data line. Each potential attacker will have a set of attack capabilities not masked by defenses, which determines his or her set of possible attacks. Finally, we consider the attackers' goals. One possible goal is to cause a denial of service. Another is to clone a system.

Now, let's examine five JTAG-based attacks. The goal here is to see the general range of possibilities, not to exhaustively list all possible scenarios. After examining these attacks in terms of the security model in Figure 2, we will be able to see what defense mechanisms we need. In a given attack scenario, the attacker possesses a limited set of capabilities:

- sniff the *TDI*/*TDO* signals,
- modify the *TDI*/*TDO* signals,
- control the *TMS* and *TCK* signals, or
- access the keys used by testers.

The system topology in all of the attacks we will present is the same as that in Figure 1. We assume throughout this article that the attacker is capable of providing or otherwise controlling the attack chip. An attacker can combine one or more attacks to achieve his or her goals.

### Sniff secret data

Figure 3 shows the sniffing attack. The attacker's goal is to learn a secret that is being sent to a victim chip via JTAG. An additional requirement for this attack is that the victim chip is downstream from the attack chip on the same JTAG chain. As Figure 3 shows, the attack chip exhibits a false BYPASS mode that is externally identical to true BYPASS mode, but which parses JTAG signals. When the secret is sent to the victim chip, the attack chip captures a copy. The secret is then delivered to the attacker either through JTAG interrogation of the attack chip in the field, or through a side channel. Alternatively, the attack chip might directly make use of the sniffed data.

### Read-out secret

In a read-out attack (see Figure 4), the attacker's goal is to learn a secret that is contained in the victim chip. We assume the attacker is capable of using I/O drivers in the upstream attack chip (attacker 1) to forcefully control the *TMS* and *TCK* lines. An additional requirement for this scenario is that the attack and victim chips are on the same JTAG chain with the victim chip sandwiched between the attack chips.

The upstream attack chip forcefully acts as the JTAG bus master, and performs a scan operation on the victim chip to access embedded secrets. The downstream attack chip collects the secret as it emerges from the *TDO* of the victim chip. This technique can be used by embedded attackers as described or by an attacker who can attach external hardware to the system under attack.

### Obtain test vectors and responses

In the test vector collection attack (see Figure 5), the attacker's goal is to learn both the vectors used

to test the victim chip and the normal responses. This attack requires that the attack chip be downstream from the victim on the same JTAG chain.

The upstream attack chip collects test vectors as they are sent to the victim chip. The downstream attack chip collects the responses as they propagate from the device under test back to the tester. Knowledge of the test vectors and responses helps an attacker further infiltrate a system by enabling the attacker to produce trojaned parts that pass normal tests.

### Modify state of authentic part

The attacker's goal is to modify the victim chip's state. We assume that the attacker can insert strong I/O drivers in the attack chip to forcefully control the *TMS* and *TCK* lines. An additional requirement for this attack is that the victim chip is downstream from the attack chip on the same JTAG chain.

The attack chip takes control of the *TMS* and *TCK* lines, and puts the JTAG test access port of the victim chip into a state where it can shift data in, thereby setting the state of registers within the victim chip, including registers that affect its normal operation.

### Return false responses to test

In a false responses attack (see Figure 6), the attacker's goal is to deceive the tester about the victim chip's true state. An additional requirement for this attack is that the victim chip be downstream from the attack chip on the same JTAG chain.

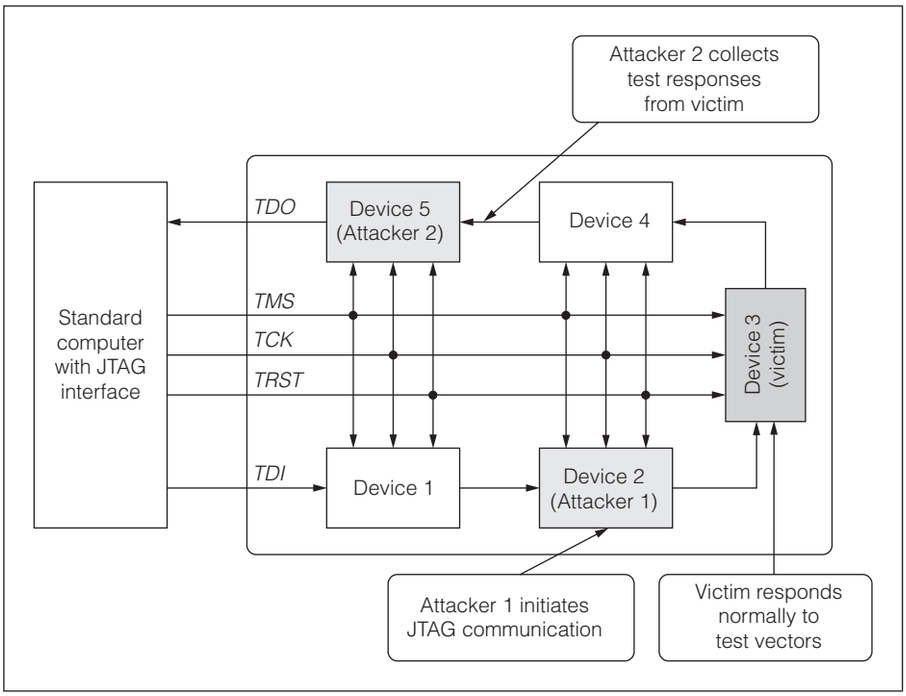


**Figure 4. In a read-out attack, the attacker obtains an embedded secret by forcing test vectors onto the JTAG lines.**

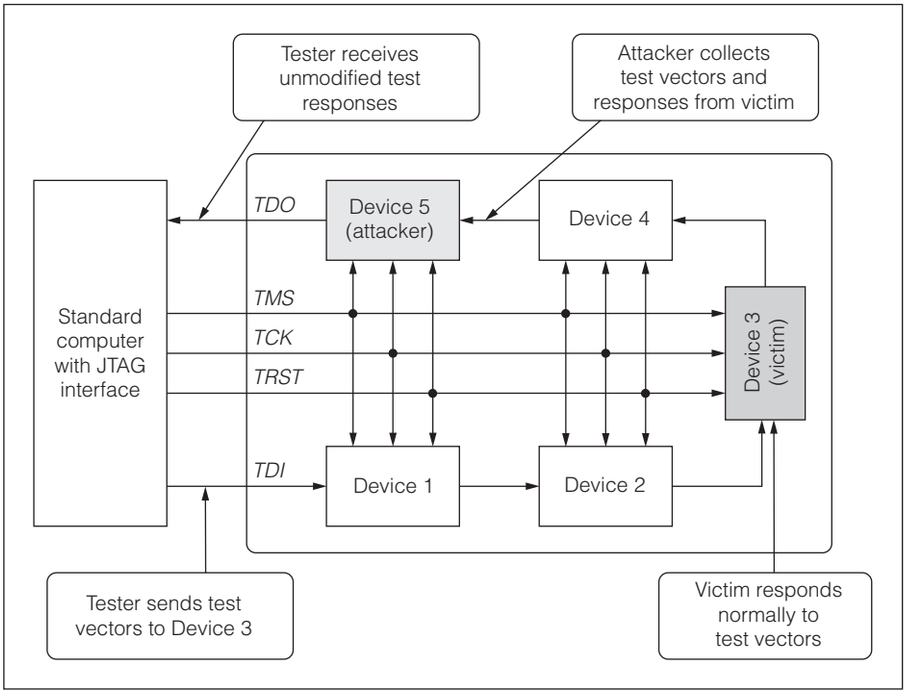


**Figure 5. In a test vector collection attack, which can be passive, the attacker obtains a copy of the test vectors and normal responses of a chip in the JTAG chain.**

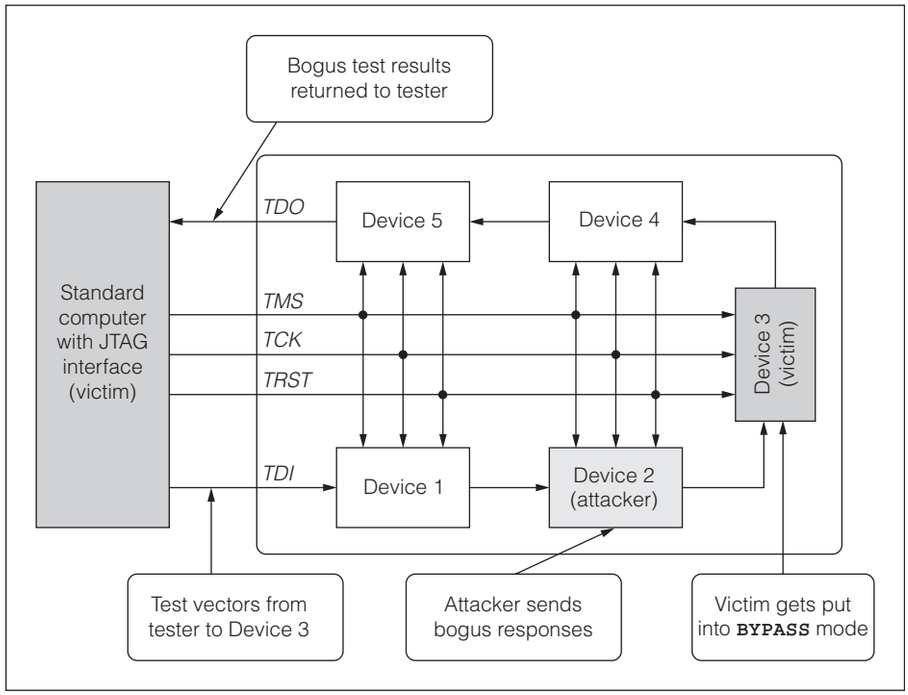


**Figure 6. The attacker can intercept test vectors that are sent to another chip, and can send false responses to the tester.**

The tester attempts to apply test vectors to the victim chip, which is not the first chip in the JTAG chain. To do this, the tester tries to place the other chips into BYPASS mode. The attack chip ignores this request and intercepts the test vectors, while instructing the victim chip and other downstream chips to enter the BYPASS mode. The attack chip can then transmit the bogus test responses back to the tester.

### Forcing TMS and TCK

Whether an attacker can forcefully control *TMS* and *TCK* depends on various factors—for example,

- strength of output drivers on the JTAG master,
- strength of output drivers of the attack chip,
- presence of buffers in the *TMS* and *TCK* lines,
- presence of series output resistor (typically 100 ohms),
- topology of the JTAG bus (star or daisy chain),
- physical layout of the JTAG bus, and
- the input logic threshold and hysteresis (if any).

For the attacker to successfully hijack the *TMS* and *TCK* lines, he or she must be able to change the voltage seen by the victim's *TMS* and *TCK* input pins. This change must be sufficient for the voltage to cross the logic threshold. If the JTAG is set up in a star topology, with separate *TMS* and *TCK* lines for each chip, this attack will be impossible.

It is common practice for system designers to exploit the economy of wiring JTAG in a simple daisy chain topology. Some designers put buffers at various points in the JTAG wiring for reasons such as noise immunity and fanout. However, these buffers add cost and complexity, and the JTAG standard doesn't require them. It is also common practice to add a resistor, typically 100 ohms, in series with each of the JTAG master's outputs. This resistor reduces ringing on the line by providing series termination and/or slowing the slew rate. In systems where this series resistor is present, there is a reduction in the amount of current that must flow in order for the hijacker to force a *TMS* bit, or to force a *TCK* edge.

The strength of the drivers in the master and in the attacker is also relevant. If the attack chip is an ASIC, the attacker can specify essentially any driver strength. If the attack chip is an FPGA in which the attacker controls the configuration bits, the attacker can program the I/O block to use a high-current I/O standard. If the attacker can control the PCB design, he can bond multiple pins of the attack chip together to form a megadriver. Finally, if the *TMS* and *TCK* lines are simply bussed around without buffers, multiple attackers can gang up to overpower the master.

**Experimental validation.** Our experimental setup to evaluate the practical feasibility of JTAG hijacking consisted of three chips: system controller was a Philips 8052 microcontroller configured transmit JTAG signals directly from its general-purpose I/O pins rather than going through an I/O expansion chip (such as an 8255) or a resistor. The victim was a Xilinx Spartan 3e FPGA; the attacker was a Spartan 3 FPGA. The 8052 was programmed to keep the *TCK* low.

The experiment tested the hypothesis that the attacker could raise the *TCK* line sufficiently to

edge-trigger the victim chip. We programmed the victim chip to count *TCK* transitions and to display the count on a set of seven-segment LEDs. This let us confirm the extent of the attacker's control over the *TCK* signal. We programmed the attack chip to send 1,000 pulses. Counting the number of *TCK* transitions received by the victim chip, we confirmed that the Spartan 3 attack chip easily overpowered the 8052 system controller. In multiple runs, the victim always showed the correct count.

When one chip overpowers the output of another chip, there is a risk that the output driver circuitry of one or both chips will be damaged. In the experiment, we minimized the chance of overheating either chip by using a *TCK* waveform with 80-ns pulses and a 0.0015% duty cycle. We left the setup running for more than an hour and saw no evidence of any damage to any of the chips.

**Common I/O driver characteristics.** Electrically, DC models can be used to represent the situation of two output drivers fighting, one driving a logic-low signal and the other driving a logic-high signal onto the same wire. Each driver will have a current-voltage (*I-V*) curve for its logic-low state and an *I-V* curve for its logic-high state. The intersection of the logic-low *I-V* curve of the 8052 and the logic-high curve of the Spartan 3 gives the voltage at the node where their outputs meet. This is similar to using load-line analysis to solve for the quiescent point of an amplifier. Using a pulse method, we obtained the *I-V* curves of the output drivers of our 8052 and our Spartan 3. From these curves, we obtained the equilibrium voltage of fighting drivers. The critical issue here is whether this voltage exceeds the high-level input voltage ($V_{IH}$) of the victim chip.

In our experimental platform, the receiving chip was a Spartan 3e using the 3.3-V CMOS I/O standard where $V_{IH}$ was 1.1 volts with no input hysteresis. The Spartan 3 easily overpowered the 8052 and could force the *TMS* and *TCK* signals. In our measurements, the pulse width was 80 ns, with a repetition period of 1.3 ms. This low duty cycle, 0.0061%, allowed hostile JTAG communication but resulted in a worst-case dissipation of only 22 $\mu$W in the output driver of either chip—not enough to cause damage.

**PCB layout effects.** Now that we've discussed the *TMS* and *TCK* hijacking in terms of DC characteristics such as *I-V* curves and logic thresholds, let's examine

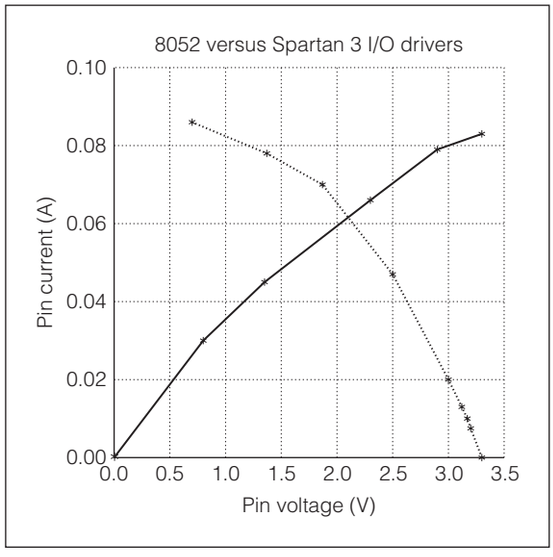


**Figure 7. A Philips 8052 (the system controller) was programmed to keep one of its pins low. The current-voltage (*I-V*) curve for this output driver was extracted using a pulsed *I-V* measurement. A Xilinx Spartan 3e was programmed to keep one of its pins high. This pin's *I-V* curve was also extracted. The result is shown. The solid line is the FPGA; the dashed line is the microcontroller. The intersection is at 2.1 V, exceeding $V_{IH}$ for most 3.3-V logic.**

a transient phenomenon that allows even weak hijackers to control the bus regardless of the master's strength.

As Figure 8 shows, a PCB trace typically connects the master's JTAG signals to each slave. The trace length from master to hijacker, $d_{MH}$, results in a round-trip wiring delay of at least

$$rtt = \frac{2d_{MH}}{v_{prop}}$$

where $v_{prop}$ is the propagation speed of a pulse in the PCB trace. This propagation speed is typically around $2 \times 10^8$ m/s. If we assume a PCB trace of 30 cm, we obtain a round-trip time of at least 3 ns. Even if the master can present a zero-ohm impedance at its end of the transmission line, each time the hijacker applies a $0 \rightarrow 1$ step to the *TMS* or *TCK* line, a pulse at least 3 ns long will appear at the victim. This is sufficient to control a JTAG TAP. We experimentally verified this phenomenon using the setup shown in Figure 8, wired together using short sections of Category (Cat) 5 cable, $Z_o = 100\ \Omega$. The hijacker can control the victim's test access port

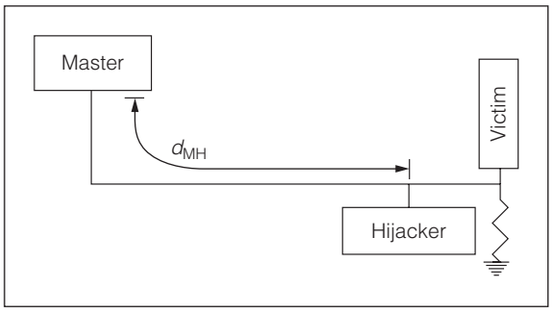


**Figure 8. A length of PCB wiring connects the hijacker to the JTAG master. This lets the attacker inject short pulses onto the wiring without being hindered by the master.**

even when the *TMS* and *TCK* lines are shorted to ground at the master's end.

## Prior work on attacks and defenses

Yang et al.[1] showed that JTAG boundary scans enable an attacker to read secret data out of a chip. They use a hardware DES implementation as their example. Satellite television receivers use trusted hardware to prevent nonpaying users from viewing the encrypted broadcasted content. An underground effort to hack satellite TV boxes by extracting their keys using JTAG is successful and ongoing.[2]

It's tempting to suggest that the deployment of JTAG is a security risk and should be excluded from trusted hardware, but untestable hardware is a prohibitive business risk. Yang et al.[3] have also proposed a DFT architecture that allows testing crypto hardware with high fault coverage yet without exposing hard-coded crypto keys to leakage. Novak and Biasizzo[4] proposed a locking mechanism for JTAG with negligible added cost by adding lock and key registers. To lock the test access port, a key is entered using a special LOCK instruction; to unlock it, a special UNLOCK instruction is issued, which requires shifting in the key via the *TDI* line. The test access port state machine is not altered. Compliance with IEEE 1149.1 is maintained except for the rejection of standard instructions (including those mandated by this standard) when the test access port is locked.

## Defenses for JTAG

Our security goals in defending against JTAG attacks are to ensure the authenticity of devices in the JTAG chain, ensure communication secrecy between the JTAG master and the chip, and reject inauthentic JTAG messages. To achieve these goals, our scheme employs three standard security primitives: a hash function, a stream cipher, and a message authentication code. With these primitives, we constructed protocols that significantly enhance JTAG security.

JTAG communication takes place between a master and a chip. The master is typically a microcontroller or a test station. The threat model of the link is similar to the threat model assumed by designers of network security protocols. Because the threats are similar, it is tempting to apply existing solutions for network communication, such as Secure Shell (SSH) and Secure Sockets Layer (SSL). Circuit complexity and key distribution are just two problems with doing this. SSH and SSL require significant amounts of computation to perform public-key cryptography. Even on the full-size computers for which the protocols were designed, the crypto-arithmetic is performed by software routines, not native opcodes. These big-integer calculations do not lend themselves to lightweight hardware implementation. Furthermore, the semantics of the JTAG standard include prompt synchronous response, so multicycle crypto-operations would be problematic.

A mechanism suitable for securing JTAG is the authentication and key establishment scheme described by Suh and Devadas.[5] This lightweight scheme uses a physically unclonable function (PUF) to authenticate the chip and to establish a cipher key for secure communication.[5] Challenge-response device authentication requires repeatability and unpredictability. Given the same challenge, the chip should always calculate the same response. Without having observed the chip's response to a given challenge, it should be impossible for an attacker to predict the response.

In our experiments with JTAG attacks, we use an idea similar to the PUF authentication and key establishment scheme, but we use fuses and a hash instead of a PUF. In a PUF, each chip's uniqueness comes from subtle physical differences between chips. For our hash, the uniqueness comes from fuse bits that are programmed at the factory.

A PUF and a hash can each support a challenge-response protocol for device authentication, and each mechanism has pros and cons. Briefly, PUFs have the advantage of being intrinsically unique, not needing uniqueness to be programmed into them. Fuses have the advantages of using less area and of being a mature technology that is reliable across temperature, aging, and power variations.

To verify the authenticity of a part in the JTAG chain, the tester sends a challenge to the chip using a custom SET_CHALLENGE instruction. The chip computes a hash of the challenge and its fuse bits. The tester collects this result using a custom GET_RESPONSE instruction. In our prototype implementation, this hash function is implemented using the Trivium stream cipher, using the input to initialize the cipher and taking the first 80 bits of keystream as the output. Strictly speaking, since the input (challenge) is fixed-length, it is not a true hash.

We also require a cipher to encrypt the communication. A block cipher is not suitable, due to its large die area. Block ciphers are also problematic in this application because they typically require multiple rounds to process each block, and the resultant delay conflicts with standard JTAG timing. The preferred mechanism for encrypting JTAG communication is a stream cipher. Although other stream ciphers could be used, in our implementation we used the Trivium stream cipher.[6]

To protect against unauthentic JTAG messages we require a message authentication code scheme. A MAC algorithm uses a key that is known to the sender and receiver, to verify that the message was sent by the authentic sender (not spoofed) and was not modified in transit. The most simplistic MAC scheme calculates the hash of the message and the key, concatenated together. This simplistic scheme is not cryptographically strong. Current popular MAC schemes such as hash-based message authentication code (HMAC; http://tools.ietf.org/html/rfc2104) use nested hash functions, each operating on a block of text. Another scheme involves the construction of MAC functions from stream ciphers in embedded platforms in which computational power is a constraint.[7]

The synchronous semantics of the JTAG protocol makes it highly desirable that the MAC verification scheme produce the answer immediately upon receipt of the final bit of the message. Introducing a delay while a message is authenticated would complicate the timing of the application of test vectors. Therefore, block-based algorithms were not desirable for our purpose. So, we used an incremental MAC function (incremental hashing could be performed so that computation took place while the message bits were being serially received[8]).

Another mechanism required for securing JTAG is access control. Access control typically involves a set of subjects, a set of objects, and at each subject-object intersection, a set of permissions. Some basic access control schemes have been proposed for JTAG, and some have been included in commodity parts. A rudimentary but strong access control mechanism is a side effect of using a MAC on the messages. If only the tester has the challenge-response pairs (CRPs) for the hash, only the legitimate tester will be able to negotiate a MAC key, so the MAC algorithm ensures that an unauthorized party will be unable to successfully issue test instructions.

### Secure JTAG communication protocol

Because of the wide range of cost and security sensitivity for different chips, a single rigid protocol will either fail to provide all the necessary security assurances to protect against attack or it will add excessive overhead to the cost of the parts. Therefore, in the security scheme we propose, we have defined four levels of protection and provide solutions for each level. Chips of varying levels can be freely mixed on the same JTAG chain without interoperability problems:

- Level 0: No assurances. This is equivalent to the vast majority of JTAG-enabled chips currently made.
- Level 1: Authenticity is assured. The authentic chip is guaranteed to be involved in the JTAG authenticity probe. No further assurances are available.
- Level 2: In addition to the assurance provided by level 1, secrecy of JTAG signals is assured.
- Level 3: In addition to the assurance provided by level 2, the JTAG link is protected against active attacks that attempt to insert or modify messages.

Table 1 summarizes these levels.

The JTAG assurance level of each chip in the JTAG chain is known by the tester. Designers can define this attribute in the boundary-scan description language (BSDL) file for the chip. Note that level 0 involves no active participation in the proposed protocol. This level can be considered an explicit provision for backward compatibility. Our scheme allows participating and nonparticipating chips to coexist in the JTAG chain, with each part functioning at its full assurance level. The presence of low-assurance chips in the chain does not affect the assurances provided by higher-assurance-level chips in the same chain. It is, however, the designer's responsibility to choose the right JTAG assurance level for a chip and its intended application.

**Table 1. Four levels of assurance and the type of security protection they offer.**

| Level | Authenticity | Secrecy | Integrity |
|---|---|---|---|
| 0 | No | No | No |
| 1 | Yes | No | No |
| 2 | Yes | Yes | No |
| 3 | Yes | Yes | Yes |

**Level 0 protocol.** Communication with a level-0 device is exactly the same as what is defined in IEEE 1149.1.

**Level 1 protocol.** Communication with a level-1 device is exactly the same as defined in IEEE 1149.1 except for the addition of an authentication operation:

1. The tester randomly extracts a CRP from storage.
2. The tester sends a challenge to the chip.
3. The chip applies the challenge to the hash module.
4. The chip sends the result to the tester.
5. The tester compares the chip's response with the CRP.
6. Regular JTAG operations commence.

**Level 2 protocol.** Communication with a level-2 device involves sending and receiving encrypted data to and from the data register (DR). The encryption affects only the signal on the *TDI* and *TDO* lines. The *TCK*, *TMS*, and *TRST* lines retain the exact semantics defined in the IEEE 1149.1 standard. The test access port controller's JTAG state machine is also unaffected. Level-2 communication occurs in two phases: setup and communication.

The purpose of the setup phase is to establish a shared secret between the tester and the chip. The setup phase proceeds as follows:

1. The tester randomly selects and extracts a CRP.
2. The tester sends a challenge to the chip.
3. The chip applies the challenge to the keyed hash module.
4. The chip uses the response as a key for the stream cipher.

In the case of the tester writing to the data register on the chip, the protocol is as follows:

1. The tester encrypts JTAG data register contents using a shared session key.
2. The tester places the chip into the SHIFT_DR state.
3. The tester shifts in the encrypted contents of the data register.
4. The chip decrypts data into the plaintext register as it is shifted in.
5. The tester places the chip into the EXIT_IR or EXIT_DR state.
6. The plaintext register is latched in and used.

The bits that are shifted out via the *TDO* during a tester-writes-to-chip operation are ciphertext. The protocol for the tester to read data from the chip is as follows:

1. The tester initiates a read operation.
2. The chip encrypts contents of the data register using the stream cipher as it transmits the bits on *TDO*.
3. The tester decrypts bits using the stream cipher.

**Level 3 protocol.** The level-3 protocol has two phases: setup and communication. The setup phase establishes the secrets shared between the tester and the chip for the cipher and MAC algorithms. The communication phase encapsulates, transmits, and de-encapsulates the contents of the data register.

The behavior at level 3 is the same as at level 2, but an additional key establishment operation is performed for the MAC key. Using that additional MAC key, the chip calculates the MAC of the bits received in one of two ways. The first is to use an incremental MAC algorithm of the form shown by Lai.[8] The second is to use a more conventional MAC scheme that requires full blocks of data before commencing MAC processing. The MAC is checked when the test access port transitions out of the SHIFT-DR state. If the MAC passes, the data register contents are transferred to a VALIDATED_DR register, and a flag on VALIDATED_DR is set, indicating that the contents of the register have been validated. Later operations can use the contents of the VALIDATED_DR register. On one hand, this approach adds area overhead. On the other hand, it enables the use of a conventional (nonincremental) MAC algorithm, the kind that has been more extensively studied.

The steps of the MAC setup phase are the same as the crypto-key setup in the level-2 protocol. The keyed hash is used along with challenge-response pairs stored by the tester. Rudimentary access control is provided by virtue of MACs being checked. Only the authentic sender can successfully negotiate a MAC key.

## Costs and benefits associated with JTAG Defenses

Successful mechanisms for enhancing the security of JTAG must satisfy cost and operational constraints, in addition to providing the required forms of assurances.

### Die area overhead

Some of the systems that we are protecting, such as mainstream consumer electronics, are high-volume commodity items that are price sensitive. JTAG security enhancements cannot add more than 10% to the die area of any protected part. We see in Figure 9 that for designs of nontrivial die area, the proposed schemes will not add significant cost. From Figure 9 we see that for a design that uses 10,000 slices, the area overhead is less than 9% even for level-3 defenses, and about 4% for level 1. On a Xilinx Spartan 3e FPGA, 10,000 slices is equivalent to 1.4 million gates.

### Test time overhead

Test time overhead is associated with the security enhancements we propose. For the level 1 defenses, the time overhead is as follows:

- The time to shift in a challenge is 80 cycles of the test clock.
- The time to initialize a stream cipher is 1,152 cycles of the functional clock. (The stream cipher we use in our reference implementation is Trivium;[6] it uses an 80-bit key and takes 1,152 cycles to initialize.)
- The time to shift out a response is 80 cycles of the test clock.

For the level-2 defenses, the time overhead is spent once per test session, to set up the cipher key and to initialize the stream cipher modules.

- Applying the authenticity verification challenge to the chip takes 80 cycles of the test clock.
- Initializing the authenticity verification stream cipher module takes 1,152 cycles of the functional clock.
- Extracting 80 bits of keystream from the authenticity verification cipher module takes 80 cycles of the functional clock.
- Initializing the communication stream cipher modules using 80 bits from the authenticity verification cipher module takes 1,152 cycles of the functional clock.

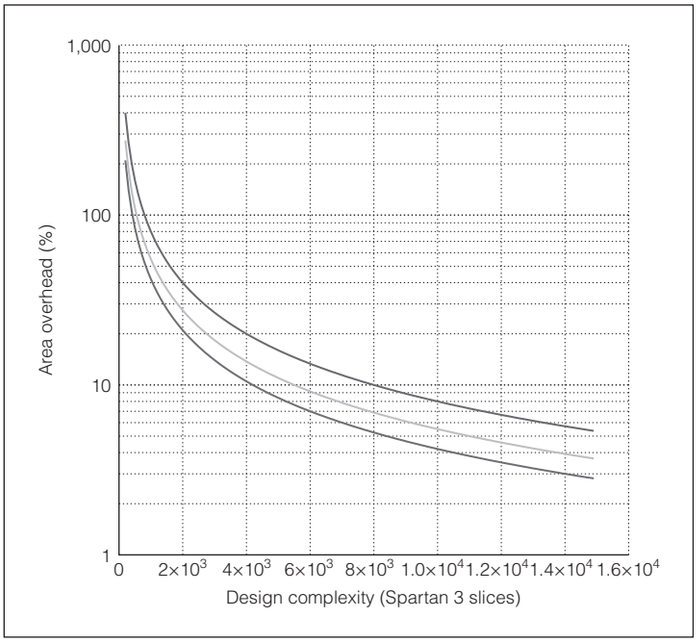


**Figure 9. Area overhead for protection levels 1 through 3, from bottom to top. The cost of the security enhancements is independent of design complexity. The percentage overhead is lower for more-complex designs. An indication of the area cost of each protection level is given by the number of additional FPGA slices used by the enhanced JTAG circuitry. These figures are for a Spartan 3e. There are no fuses in the FPGA, so fuses are modeled as hard-coded bit vectors. Overhead in an ASIC will be less.**

After this setup, there is no test time overhead associated with the level-2 defenses. The communication stream cipher module operates synchronously with the test clock to encrypt and decrypt data. For level-3 defenses, the minimum additional test time overhead is the same as for level 2.

### Operational costs

The security enhancements that we propose require that challenge-response pairs be extracted at manufacturing test time before the chips leave the manufacturing facility. If fuses are used rather than a PUF, they should be blown with a random pattern before the extraction of challenge-response pairs. For each chip that the customer receives, the challenge-response pairs are needed, which must be delivered to the customer over a secure channel.

It is also possible to eliminate the burden of keeping track of the identity of each chip on a reel of parts. Instead of querying the chipmaker for the challenge-response pairs for a part, the chip can contain a

unique code that can then be used to retrieve the challenge response pairs from the chipmaker.

### Impact of defenses on known threats

Several steps can be taken to prevent two otherwise practical attacks.

**Attacker obtains embedded secrets.** The attacker could either sniff the JTAG link as the secret (e.g., key) is being programmed or could perform an unauthorized debug session to read out the secret. These attacks are popular in hacking satellite TV boxes to extract the box keys using JTAG and in taking a snapshot of the firmware in the box.[3,9] To prevent unauthorized JTAG access, Novak and Biasizzo proposed an access control solution in which a key must be presented to the chip before the chip's JTAG test access port can be controlled.[4] Under an expansive threat model, in which hostile chips could be present in the JTAG chain, the scheme is not effective because the key can be sniffed by an attack chip while it is being sent. On the other hand, the proposed level-2 and level-3 defenses are comprehensive and guard against passive sniffing by encrypting the link. They also prevent unauthorized debug as a side effect of applying a MAC to the data.

**Attacker clones an existing system.** Cloning would involve different attacks, depending on a system's implementation style. If it is a CPU-based system or a programmable logic device–based system, cloning would involve reading out firmware or a bitstream. The attacks are similar to obtaining an embedded secret, and the defenses are the same. For example, at levels 2 and 3, the attacker would need to negotiate keys with the chip to communicate over JTAG, and this would be impossible for the attacker because he or she would lack the challenge-response pairs that were set up offline.

**In systems in which** sensitive data is transported or accessible via JTAG, we do not recommend using a daisy-chain topology with conventional unprotected JTAG. A star topology protects against many of the attacks we've discussed; however, it also increases PCB complexity and cost, and still it doesn't eliminate all JTAG-based system vulnerabilities. An alternative is to retain the daisy-chain topology, and to derive the needed security from cryptographic enhancements built on top of JTAG.

Our scheme provides a significant improvement in JTAG security with reasonable added cost. The scheme is flexible in the sense that it can provide high assurance for important chips and lower assurance (and lower cost) for less-important chips. Compatibility across the assurance levels is maintained, and compatibility with IEEE Std. 1149.1 JTAG is maintained to the maximum extent possible.

Although the work we have described here has been restricted to JTAG, a multitude of other threats to hardware exist that do not involve JTAG. For instance, exploits via bus probing have been successfully executed in the past and will likely be used again. It should be noted, however, that the original impetus behind JTAG was to provide access to I/O pins because it was difficult to probe the wiring on modern PCBs. Securing JTAG denies the attacker what might otherwise be an easy way of probing a bus. We did not address several other threats, including that of manufacturers inserting hostile functionality in chips that they supply. Detecting the presence of such functionality is an active research topic.[10] ■

## Acknowledgments

This work was supported by National Science Foundation grant 0621856. We also thank Linda Lieu for her valuable help with the project.

## ■ References

1. B. Yang, K. Wu, and R. Karri, "Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard," *Proc. Int'l Test Conf.* (ITC 04), IEEE CS Press, 2004, pp. 339-344.
2. "Dish Net: In House Made with a Locking Script"; http://www.satcardsrus.com/dish_net%203m.htm.
3. B. Yang, R. Karri, and K. Wu, "Secure Scan: A Design-for-Test Architecture for Crypto Chips," *Proc. 42nd Design Automation Conf.* (DAC 05), ACM Press, 2005, pp. 135-140.
4. F. Novak and A. Biasizzo, "Security Extension for IEEE Std 1149.1," *J. Electronic Testing,* vol. 22, no. 3, 2006, pp. 301-303.
5. G. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," *Proc. 44th Design Automation Conf.* (DAC 07), ACM Press, 2007, pp. 9-14.
6. C.D. Canniere and B. Preneel, *Trivium Specifications,* ECRYPT Stream Cipher Project, 2006.

7. B. Arazi, "Message Authentication in Computationally Constrained Environments," *IEEE Trans. Mobile Computing,* vol. 8, no. 7, 2009, pp. 968-974.
8. X. Lai, R. Rueppel, and J. Woollven, "A Fast Cryptographic Check-Sum Algorithm Based on Stream Ciphers," *Advances in Cryptology-AusCrypt,* Springer-Verlag, 1992, pp. 339-348.
9. Fwaggle, "Howto: JTAG Interface on a Dish 3700 Receiver," http://www.hungryhacker.com/articles/misc/dish3700_jtag.
10. Y. Jin and Y. Makris, "Hardware Trojan Detection Using Path Delay Fingerprint," *Proc. IEEE Int'l Workshop Hardware-Oriented Security and Trust* (HOST 08), IEEE CS Press, 2008, pp. 51-57.

**Kurt Rosenfeld** is a doctoral candidate in computer science at Polytechnic Institute of New York University. His technical interests include electronics, security, and instrumentation. He has an MS in electrical engineering from City University of New York. He is a student member of the ACM.

**Ramesh Karri** is an associate professor of electrical and computer engineering at the Polytechnic Institute of New York University. His research interests include secure hardware design, architectures for network protocols and encryption, and fault-tolerant nanoscale systems. He has a PhD in computer science from the University of California, San Diego.

■ Direct questions and comments about this article to Kurt Rosenfeld, NYU-Poly, 6 Metrotech Plaza, Brooklyn, NY 11201; kurt@isis.poly.edu.