SAND2016-0686C

Secure Embedded System Design Methodologies for Military Cryptographic Systems

Gary N. McGovney and Deborah L. Jensen
Sandia National Laboratories
High Integrity Software Systems
PO Box 5800
Mail Stop 0860
Albuquerque, NM 87185

Abstract: *Cryptographic embedded systems are used to safeguard both access to classified data and controls for external subsystems. The assurance required for these systems extends beyond simply denying access to unauthorized users, hardware failures must not lead to the inadvertent activation of protected subcomponents or allow access to classified data. As more systems are transitioning from fixed hardware and state machine designs, for which more formalized failure type and rate calculations exist, there is an increasing need for this level of assurance for software-based processor systems in the absence of equivalent failure type and rate calculations for software. This paper presents a method to guarantee to a quantifiable level of confidence that external components are driven if-and-only-if an authenticated request is received. This is accomplished through the use of software algorithms, data storage formats, and analyzable comparator hardware detection and blocking logic.*

Keywords: Secure Embedded System Design; Fail-Safe Design Assurance; Secure Software; Fault-Tree Analysis (FTA); Built-In Self-Test (BIST)

Introduction

Secure access-control systems restrict operations to authorized users via methods as simple as a password or as complex as encrypted messages with digital signatures. In the case of systems which limit access to or control of external components, the concern is not only that access be denied to unauthorized users, but that operations cannot occur autonomously even under failure conditions. In software-based processor systems that control external interfaces with access granted or denied based upon password validation, there are hypothetical failures in which, through a number of mechanisms, program execution could transfer to the output functions in the absence of a valid password.

While this is very unlikely, particularly in a dual-redundant processor comparator design which would require common-mode failures in the individual software/processor elements, the question of exactly how unlikely is difficult to answer. Fault-Tree Analysis (FTA) has a wealth of information backing probability, types, and behavior of hardware failures – but far less for software. We present a method that removes software error calculations from FTA by using the analyzed, validated, and tested comparator hardware circuitry as the final arbiter for output operations. That is to say, the probability of software failure can be assumed to be 1 and the probability of successfully driving outputs in the absence of a valid password can be made as small as desired by appropriate selection of comparator bit-width and signal complexity.

We do this by removing all knowledge from the software of how to control the outputs correctly (including how many outputs are driven, which outputs are driven, assertion/de-assertion order, etc.) and instead use a set of output register patterns that the software output function dutifully writes to the comparator output register until the data is exhausted.

Furthermore, the data used by the software output function is not stored as plaintext, it is covered by a logical XOR function with a secure hash of the password (not stored in the system) and a data phrase unique to each processor (stored in the system). In a properly functioning system this output function will never be called unless a valid password is received and authenticated. In the case of a hypothetical failure that attempts to operate without a password, or with an invalid password, the data to correctly control the outputs cannot be generated. Not only that, the stored data is dramatically different in each of the two processors which will guarantee comparator mismatch and alarm lockup if the output function is accidentally initiated.

Comparator Design

The use of dual-redundant processor comparator designs in security-critical applications, such as those requiring NSA Type 1 cryptographic certification, is certainly not new, but there are differences in implementations. In order to meet Fail-Safe Design Assurance (FDSA) requirements in which systems secure data against multiple failures, comparators must have redundant parallel mismatch detection, redundant serial data blocking, and regular self-testing which verifies all detection, alarm, and blocking circuitry.

Figure 1 shows an example of mismatch detection and blocking circuitry logic with a single bit output as an example. The two XNOR gates act as the parallel compare mismatch detectors. Having two mismatch detectors in parallel prevents a single fault from hiding a bit error. The two AND gates act as the redundant serial blocking

element. Having two blocking gates prevents a single fault from allowing mismatched data to pass. In practice, comparator designs will have multiple n-bit wide interfaces for communication and peripheral I/O.

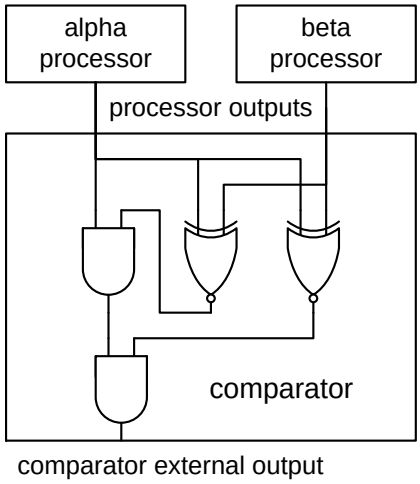

Figure 1. Basic dual-redundant processor comparator architecture

A self-test capability is required to guard against the gradual accumulation of errors degrading the designed and validated level of fault protection. Figure 1 does not show the hardware Built-In Self-Test (BIST) which executes at power-on, only releasing the processors from reset upon successful test completion. The BIST should inject test signals at every compare input (XNOR gates) and monitor every blocking gate output (AND gates) to independently verify that each element is functioning correctly. Complete and correct execution of each self-test phase should generate a unique checksum which is not stored in the comparator and that software validates upon release from reset. We feed the test inputs and outputs through a Linear Feedback Shift Register (LFSR) to generate this value which is checked by both the BIST hardware and the processor software to validate test completion. The software check guards against hardware BIST functional failures and completes the hardware-checking-software/software-checking-hardware synergy that is an essential design philosophy in secure embedded systems.

Alarm circuitry is also not shown in these simplified diagrams, but external access must be prevented on the first sign of erratic behavior. In addition to blocking output, mismatch detection must latch an alarm state that 1) clears all output registers and 2) cannot be cleared by either processor. A system reset that forces a comparator hardware BIST and processor initialization is the best option to avoid any possibility of indefinite thrashing on the processor output registers until a match occurs.

The method of synchronizing the dual processors (transparent synchronization via bus locking or slow-peripheral wait states, handshake-based interlocks, etc.) is not important for this discussion, what is important is that the external outputs be implemented as a parallel write or at least compared as a single parallel operation. Increasing the number of output signals will result in an increased level of confidence that random activity will not only fail to properly control the outputs but will alarm and lock the system. This alarm will occur regardless of whether the outputs are actually used externally, or are simply there to force additional bit-compare operations which increase the likelihood of a mismatch during invalid operations.

The diagram in Figure 2 shows a basic comparator configuration. When both the alpha and beta processors attempt a write to the outputs, one of two actions will occur. If every bit in each processor register is identical, the external outputs will be latched to the new state; if instead even a single-bit mismatch occurs, the comparator will alarm, clear all outputs, and require a system reset to recover.

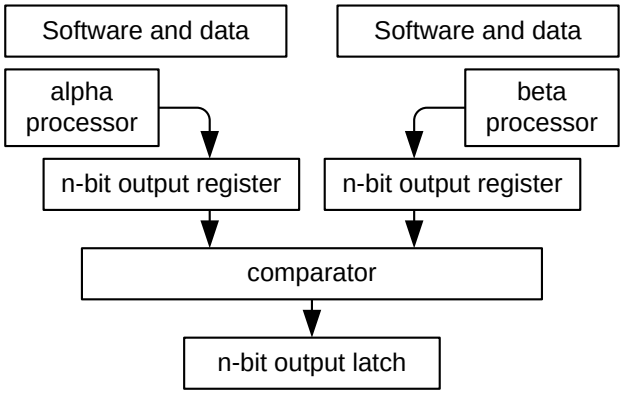

Figure 2. Comparator general purpose output configuration

Knowledge Separation

As a simplified example of a protected external signal, we present a device which sends a single transition to output 0, changing from the initial "0" off state to the "1" on state, in a system with a 16-bit wide comparator output register. There is no knowledge that output 0 is the correct line or that the line must be driven from low to high in the software algorithm. The output could depend upon any number of other lines driving high or low. For this simple example there is only one 16-bit output value to write to the general purpose output register, 0x0001 (with output 0 as the least significant bit).

The plaintext value 0x0001 is not stored in either processor. Instead, as shown in Figure 3, it has been covered by logical XOR with the hash (SHA-256 in the example) of the valid password ("Laconic" in the example) concatenated with a processor-specific constant ("-alpha" and "-beta" in the example) prior to storage in each processor's private memory. Note that only the first 16 bits of the hash value

are used. The value created for processor alpha is 0x72FE and the value created for processor beta is 0xF770.

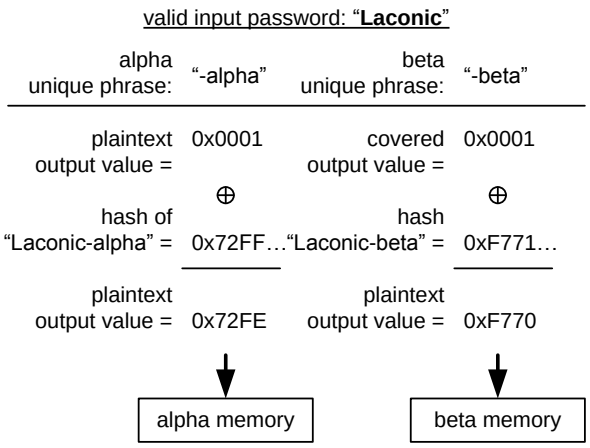

Figure 3. Creating covered output data for each processor

The output algorithm in each processor, after having received and validated the correct password, performs the following operations (shown in Figure 4) to reveal the plaintext output value and assert the external signal:

1. Concatenate the provided password with the processor-specific constant and hash the concatenation
2. XOR the first 16 bits of the hash value with the covered data to reveal the plaintext value
3. Write all output values (only one in this example)

Obviously this approach is readily extensible to more complex signal requirements with multiple outputs and complex waveforms or even communication protocols. The data fields can be given more complex functions than simple output register write values (such as mask values to protect specific output lines from changing state during a given write operation) in order to accommodate systems driving multiple independent outputs attached to the same output register.

Failure Calculations

Should a fault condition cause the software to begin directly driving the outputs without uncovering the plaintext output values, or to uncover the values with an incorrect password, the software will either write the covered data values to the comparator or the covered data XORed with the hash of an invalid password and processor-unique string. Fail-safe operation at this point depends upon the output of the two processors mismatching.

For cryptographically-strong hash algorithms, such as SHA-256 in the example, the probability of two output bit positions having the same value for different input strings is effectively ½. Each output bit is considered independently in the comparator prior to output and even a single bit failure will alarm and lock down the system - there are no second chances with the comparator.

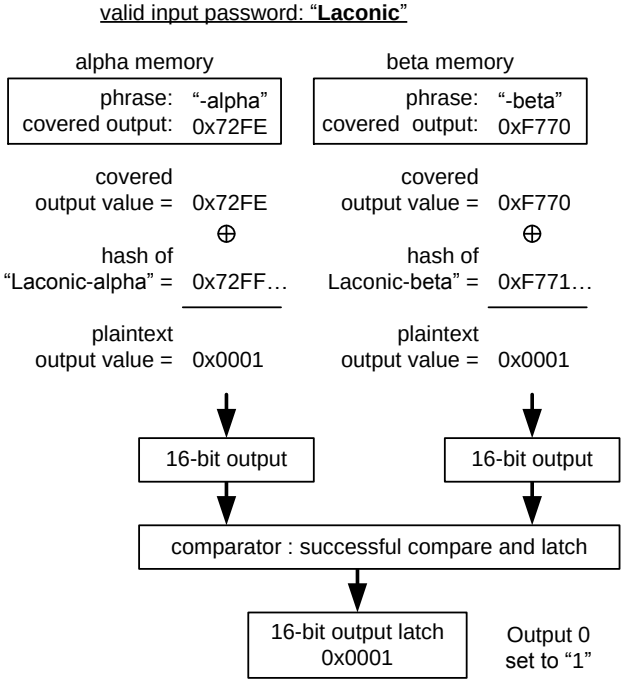

Figure 4. Uncovering and using output data in each processor

Avoiding an alarm condition with an exact match between the processors in this 16-bit example is equivalent to tossing a coin 16 times and coming up heads each time. Figure 5 shows the probability that random data will be caught (the comparator will alarm) for the 16-bit example described as well as for a system with a 32-bit wide output register.

$$16-\text{bit output width calculation:}$$
$$1-\left(\frac{1}{2}\right)^{16}=1-1.526\cdot 10^{-5}=0.99998$$

$$32-\text{bit output width calculation:}$$
$$1-\left(\frac{1}{2}\right)^{32}=1-2.328\cdot 10^{-10}=0.9999999998$$

Figure 5. Probability that invalid operations will be detected

There are a couple of conservative simplifications in these calculations. In order to control the external system without a valid password the random data must not only match in each processor to make it through the comparator, it must also create the required signal. For example, while a resulting value of 0x0000 in each processor would make

it through the comparator without alarming, it would not create the correct signal to control the system. An alignment of random matching data with actual control signals becomes even more unlikely for systems that require more than one output line.

Also, we have chosen the most trivial signal possible - one with a single state change. Adding a second required state change doubles the number of bit matches required and has the same effect as seen with doubling the register bit-width in Figure 5.

Conclusion

It is possible to prevent software from driving external signals without authorization to a quantifiable level of confidence using a simple, secure comparator design with a parallel interface for general purpose output control, an algorithm which writes all of the outputs at once based blindly upon data rather than knowledge of which outputs are of importance, and control data unique to each processor which is covered by a hash dependent upon authorization data that is not stored in the system. This is done without knowledge or dependence upon software failure rates or mechanisms and requires only an analyzable hardware design.

Acknowledgements

The authors wish to thank Elmer Collins of Sandia National Laboratories for years of sharing his extensive knowledge of Fail-Safe Design Assurance and Fault-Tree Analysis. Countless discussions in which he shared insights into potential error mechanisms and vulnerabilities, and acted as a sounding board for our proposals, have been invaluable in shaping our approach methodologies toward secure embedded system design.