Lecture Notes

# Chapter 9:
# Hardware-Based Security

CYENG 351: Embedded Secure Networking

Instructor: Dr. Shayan (Sean) Taheri
Gannon University (GU)

# Chapter Overview - High Performance in Silicon

- What are hardware alternatives that allow embedded developers to get the most out of their systems?
- This chapter presents the topic of hardware assistance.
- With the advent of the Internet, and the increasing need for security, embedded CPU manufacturers have started adding special instructions to their chips in order to help speed up software implementations of cryptographic algorithms.
- It discusses some examples of instructions that are useful for common algorithms, as well as some real-world examples of embedded processors that provide this added functionality.
- It introduces the features of these processors, and looks at a strategy for determining which features are important for what types of applications.
- Hardware assist provides the software engineer with hardware resources to handle some of the more expensive cryptographic tasks, but how are these instructions and resources used?
- It shows some sample code of software implementations that utilize these resources on real-world processors, and discuss how an engineer can go about developing his or her own optimized implementations.
- It explains assembly programming and compiler-supported optimizations, as well as out-of-the box solutions.
- It presents complete hardware security solutions.
- It introduces complete IC packages such as the Atmel Trusted Computing Platform, and components that can be added to an FPGA device or processor to provide complete security support in hardware.
- It reviews some other hardware features, such as write-once key registers and physical protection mechanisms.
- It discusses on-chip versus off-chip solutions.
- There are advantages and disadvantages to each, such as on-chip solutions preventing bus monitoring, but using up valuable CPU silicon.

## Hardware Assistance - Specialized Instructions/Mechanisms

- The concept of hardware assistance for math-intensive operations is as old as the modern computer.
- Today’s processors now include instructions that speed up floating point calculations, something that used to be included in expensive co-processors.
- We still implement operations in hardware that could be done in software but benefit from the enormous performance gain.
- These operations do not make up the entire operation, but rather speed up some common operation that would take much longer in software.
- One of the primary examples of hardware assistance for cryptography in a CPU is the inclusion of an instruction or instructions that perform repeated operations in hardware, rather than relying on some software mechanism.
- As an example, the RSA public-key encryption algorithm consists of raising a large number to a large power with a modulus.
- Thus, the operation is essentially an enormous number of multiplications of large numbers.
- The final result and all intermediate results are limited to the size of the modulus, so we don’t have to worry about memory, but the number of multiplications (in RSA this would be determined either by the public exponent or the private key) is bounded by the number that can be represented by the key size—for even 512-bit RSA, this can be millions of operations (primarily with the private key).
- As a result, the RSA operation can end up being very slow. Compounding the issue is the fact that the numbers we are dealing with in RSA operations are not supported by the native instruction set of general-purpose processors - the RSA numbers are many bytes long (512 bits = a 64 byte integer!).
- Since the largest numbers supported by today’s processors max out at 64 or 128 bits, any RSA software implementation must break up the multiplication operations on these long numbers into a number of smaller operations.

## Hardware Assistance - Specialized Instructions/Mechanisms (Cont.)

- To speed up RSA, we could implement the entire operation completely in hardware, but this uses valuable silicon, so a less-intensive operation may be desirable.
- One thing that is extremely common in RSA is the multiplication of two large numbers (remember, 64 bytes—minimum!).
- Splitting that operation up into pieces that can fit into registers will take a lot of instructions (or a lot of repetition of a few instructions, anyway), and the instruction fetch alone will start to impact the end performance.
- At speeds in the tens or hundreds of megahertz (at which many embedded processors run at today), the combined effect can lead to a delay of several seconds - often too long, especially if you are waiting for a web page to load.
- The key is to provide instructions that can do the repetitive work for you.
- For RSA, one of the most obvious bottlenecks is the multiplication operation. If implemented in software it requires that the numbers be loaded into registers a few bytes at a time (depending on the processor register size and instruction set) and multiplied appropriately, a piece at a time.
- Then, any carry value from the multiplication needs to be saved temporarily while the result is stored and the next pieces of the numbers are loaded.

## Hardware Assistance - Specialized Instructions/Mechanisms (Cont.)

- What if the processor had an instruction that did exactly that and repeated as many times as you wanted?
- This is obviously a CISC-style instruction, since it would take a variable number of CPU clock cycles to execute, but it could be implemented as a peripheral device if the CPU was RISC-based.
- In any case condensing the load, store, multiplication, and addition operations into a single load-once-and-go instruction can result in a speed up of 10 times or more!
- The reason for this is that you have eliminated the additional memory accesses that would be required to reload each of the instructions that would otherwise be required.
- An additional bonus is that the intermediate results are handled internally.
- Your program would no longer have to keep track of the temporary information between operations.
- The real reason to look at hardware assistance as opposed to full-blown hardware implementation of your algorithm is to avoid using valuable silicon, but it also means that you can find operations that are useful for purposes other than for what they are primarily intended.
- This also means that if for some reason your encryption algorithm needs to change later, there is a better chance that the general-purpose hardware will be reusable, whereas if you had implemented that algorithm entirely in hardware, you would be stuck with it.

## Hardware Assistance - Specialized Instructions/Mechanisms (Cont.)

- Note that the hardware assistance does not necessarily need to be implanted in the same silicon as the CPU, it would be reasonable and nearly as fast (or faster in some cases) to implement the hardware-assist logic as a peripheral.
- For this to work, the hardware would be implemented the same as any another peripheral in the system, with some interface accessible to the CPU core (the interface could be some type of control registers or direct memory mapping of the peripheral).
- In this case, the expensive (in clock cycles) operation would be offloaded from the CPU itself. The CPU would copy any appropriate information to the peripheral and then initiate the operation.
- The peripheral could then fire an interrupt whenever it completed the operation, thus freeing the processor to do other work.
- A simpler and perhaps less dangerous implementation would have the CPU simply wait for the result of the operation.
- In any case, the peripheral option would keep the instruction set smaller or allow for a RISC-style instruction set while still providing the performance gain of hardware assistance.
- For a real-world example of hardware assistance in action, we will look at the Rabbit 4000 microprocessor.
- The Rabbit, originally based on the venerable Z80 instruction set, has evolved into a more versatile processor, and includes many options not found in its ancestors.
- One of the notable features is the inclusion of some hardware-assistance instructions for cryptographic operations.
- One of these is the so-called “sbox” instruction, which provides access to an sbox, a lookup table used by AES implementations.
- The lookup table would otherwise have to be implemented as an array, slowing the AES algorithm considerably.

## Hardware Assistance - Specialized Instructions/Mechanisms (Cont.)

- Other notable cryptographic hardware-assist features of the Rabbit 4000 are the unsigned multiply-and-add (or subtract) instructions, known as UMA (and UMS).
- These instructions enable a greater than 10 times speedup in RSA operations over a full-software implementation using the other Rabbit instructions.
- The UMA instruction, specifically, is designed to make the operation of adding two arbitrarily large unsigned integers very fast.
- Basically a block-copy instruction with some additional logic, UMA can be used to make cryptographic operations much faster, particularly those operations used by the RSA algorithm.
- The speedup from the UMA instruction is significant enough to allow the Rabbit, an 8-bitter running at less than 60 MHz, to perform an RSA decryption quickly enough for use in HTTPS (HTTP over SSL) for web-based interfaces (about 2 seconds for 512-bit RSA).
- Hardware assistance is great for providing a level of security to a class of hardware not normally capable of the higher-level processing required by modern cryptography, but if you want to be serious about security, you need to spend a lot more money on a more powerful processor or a little bit more on a specific hardware solution.

## Hardware-Based Solutions

- Modern PCs are so fast that software solutions are usually acceptable (less than a half-second for an RSA operation is plenty fast for most applications).
- In the world of embedded devices with limited resources, implementing a full SSL stack in software may not be easy.
- Hardware assistance can get you part of the way to a secure system, but there are still limitations.
- For embedded applications where you need high speed and high security, you have two options: Go for a bigger, faster CPU or implement the security in hardware.
- One very feasible option for a hardware-based security solution would be to implement an SSL stack or cryptographic algorithms on an FPGA.
- This has the benefits of being customizable, and the cost of an FPGA can be less than a more powerful CPU (obviously depends on what you are doing).
- Most likely, you would want to implement the cryptographic algorithms for the FPGA and leave the full SSL stack in software.
- Using an FPGA should result in a fair gain in performance, but it will not be as fast as something forged in silicon.

## Hardware-Based Solutions (Cont.)

- Fortunately for us, there are a few vendors that offer security-on-a-chip solutions.
- The idea of putting security algorithms in silicon is not a new one.
- When RSA was first developed, there was an effort to design a chip dedicated to the algorithm.
- While the idea is less relevant for mainstream computing now that PCs have processing power that was unthinkable in the 1980s, the concept of putting cryptography into hardware still makes sense for embedded applications.
- For the types of applications we have been discussing, it is almost a necessity in certain cases.
- A real-world example of a security-on-a-chip solution is Atmel’s Trusted Platform Module, which provides a full RSA accelerator (2048-bit RSA in 0.5 seconds) along with a random number generator and nonvolatile memory for key storage.
- At only a few dollars each, this is a serious alternative to using a faster processor.
- With the additional key storage and random number generator, it would be a useful addition even to an application that used something more powerful.
- Other vendors offer similar technologies, but they are not extremely common due to a lack of need.
- When most processors (even the embedded ones) can do an RSA operation in less than a couple seconds, there really isn’t much need for hardware acceleration except under the most extreme circumstances.

## Hardware-Based Solutions (Cont.)

- Security is truly dependent on the application. Adding another chip to your design may not be practical, and it comes with its own problems.
- One of the major issues is actually physical security, which we will talk a little bit more about in the next chapter.
- The issue is that you are placing a chip somewhere else on a circuit board, and there needs to be an interface to that chip.
- If your device is going to be deployed out in a field somewhere, what is there to stop someone with a logic analyzer from tapping into that interface and grabbing information?
- If the keys are stored in memory and transferred to the processor, the data bus is also a target.
- With the relative scarcity of security hardware solutions that fit the bill for limited-resource embedded applications, you are probably better off just living with the performance hit of a software implementation or just pay more for a faster CPU.
- Another issue is that security is a fast-changing field, and if you are locked into a particular hardware-based solution, it gives you very little room to adapt your application.

## Assignment

- **Reading Assignment:**
  - Stapko, T., 2011. **Practical embedded security: building secure resource-constrained systems**. Elsevier.
    - ✓ "Chapter 9: High Performance in Silicon", Pages 173-178.

# Questions?