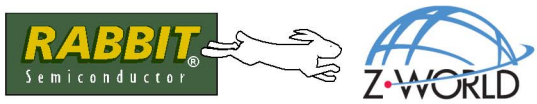
White Paper



WP106

Cryptography for Engineers Who Couldn't Care Less

by **Jim Turley,**
Microprocessor Analyst and Editor-in-Chief,
Embedded Systems Programming

Encryption and digital cryptography are profoundly uninteresting to the average engineer. Whether you're a programmer or a hardware developer, straight out of school or wizened by years of experience, you probably couldn't care less about implementing crypto in your current project. But you may not have a choice.

Encryption and cryptography mean different things in different applications. You can scramble all network traffic and e-mail, for instance. Or just add security to specific Web transactions, such as when you're using a credit card for an on-line purchase. Then there's non-Internet-related encryption, such as what all DVD players have now and what MP3 players and stereo gear are likely to get. Content protection and digital rights management (DRM) rely on encryption to protect copyrighted material. Even car stereos will soon include encryption to secure the digital data flowing through their optical in-car networks. The point is, most systems are going to require encryption, digital rights management, secure sockets, or some other form of cryptography. Like it or not, we're all about to become spooks.

Serious cryptography is scary stuff. Even DSP programmers or experts in arcane network topologies think crypto guys are strange. Fortunately, we don't all have to become crypto experts in order to build basic security features into our products. All it takes is the right prepackaged hardware and/or software.

Foiling a Supercomputer with an 8-Bit Processor

First, some basic terminology. “Cryptography” covers a lot of territory, including traditional, non-electronic things like secret decoder rings and invisible ink. Within the digital realm, there are a number of ways to obscure data. Everyone has their favorite method, but a number of standard “canned” encryption algorithms have emerged. Oddly, publishing the details of a standard encryption algorithm doesn’t diminish its security; many codes are just as hard to break whether you know how they work or not.

“Like it or not we’re all about to become spooks”

Some of the most common data-scrambling algorithms go by lyrical names like DES (which simply stands for data-encryption standard), 3DES (also called triple-DES), AES (advanced encryption standard), ARC4 (alleged Rivest cipher four), SHA-1 (secure hashing algorithm), and others. The mathematics behind all these standards is inscrutable, but that’s okay because we don’t need to understand specifically how they work. Like logic gates, it’s enough to know that they *do* work, and move on to making something useful from them.

Cryptography would seem, by its nature, to require powerful computers to encrypt or decrypt messages. After all, we hear about how certain encoding methods will take a supercomputer 500 years to crack. If that’s the case, how can we possibly encode data with inexpensive chips and software? The key (ahem) lies in the fact that data-encryption standards are like padlocks: they’re trivially easy if you have the right key, but devilishly hard if you don’t. These five-century cracking claims are for supercomputers that *don’t* have the right key. If you *do* have the right key, a simple 8-bit microcontroller is enough. Well, a microcontroller and a few milliseconds.

Put Data in Blender, Hit Purée

Broadly speaking, all of the common standards mentioned above work by swizzling data bits around in various unusual patterns. That is, they take the message data and the key data, and perform normal exclusive-OR (XOR) functions on them. Oftentimes bit fields are swapped or rotated, and a little arithmetic (ADD, SUB) might be thrown in for good measure.

Figure 1 shows how this might work. The 32-bit value, A, is combined with the 32-bit key, K, to produce an encrypted value, X. In the figure, the three topmost bits of A are swapped with another three bits lower down, and vice versa. The result is then exclusive-OR'd with the key data.

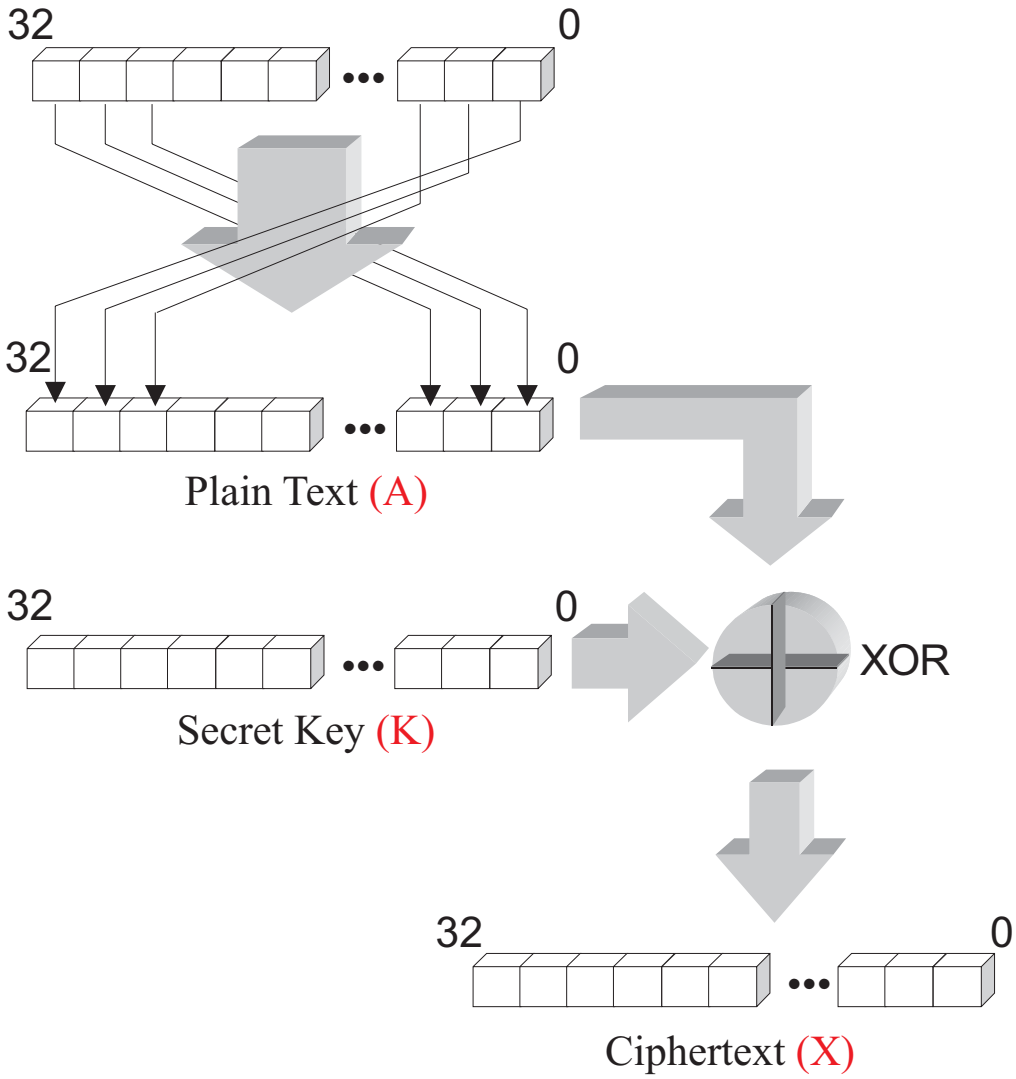


Figure 1. Encryption in Action

The "strength" of a cryptography algorithm (its resistance to cracking without the proper key) is often a function of the key data. A 128-bit key is stronger than a 64-bit key, for example. Reduced *in absurdum*, a 1-bit key would be trivial to crack because you'd need to try only two combinations. Either 0 or 1 is bound to work.

As you can see, data and bit fields are moved from all over the original message; a few bits here, a few bits there. The process is also iterative. In our example, the message data and the key data were manipulated only twice but in reality they'd be combined several times. The more lengthy and convoluted the interaction, the stronger the encryption. The strongest algorithms are prolonged and tortuous affairs.

"Encryption algorithms are often written assuming 32-bit values and 32-bit processors"

You can also see that the data are not handled in convenient byte-sized chunks. Odd-sized bit fields are pulled from inconveniently misaligned boundaries. All of the current standards assume that messages and keys are at least 32 bits long. The wireless 802.11 WEP (wired-equivalent privacy) key is 64 bits long. Most encryption standards are written assuming 32-bit values for logical and arithmetic operations, in part because 32 bits provides enough flexibility for the awkward and abstruse operations on which they rely. So a 32-bit processor must be required to handle any realistic crypto work, right? Not necessarily.

Rabbit Roundup

Data encryption doesn't require a 32-bit processor, *per se*. But it does need a processor that can elegantly handle 32-bit quantities. That presents you, as the engineer, with an economic problem as well as a technical hurdle. Big 32-bit processors are expensive, use a lot of power, put out a lot of heat, and are physically large. At the other extreme, 8-bit processors are cheap, plentiful, small – and underpowered. Caressing large encryption keys with a tiny 8-bit processor would be a painful programming exercise. Like the supercomputers, it might take 500 centuries for the average 8051 or 6805 to handle 3DES encryption, even *with* the key.

Enter the Rabbit 4000 processor. Although it's nominally an 8-bit CPU, the 4000 has a 32-bit register set that handily massages large encryption keys. Not incidentally, the chip also has dozens of new instructions specifically to help with encryption.

The 4000 is part of Rabbit's third-generation microprocessor family. It follows the Rabbit 2000 and Rabbit 3000, one of the more successful branches from the fertile Z80 family tree. Although all these chips bear strong similarities to the Z80, they're considerably more capable than their heritage would imply. The 4000's on-chip DMA controller and integrated 10Base-T Ethernet MAC and PHY are just icing on the cake.

"Although it fetches 8-bit instructions the Rabbit 4000 handles 32-bit operands"

For starters, the Rabbit 4000 can run at 100 MHz, or about 10 times faster than other 8-bit processors. It can address an enormous 16 MB range of memory, as much as the 32-bit Motorola 68010 processor. The chip's data bus can operate in 16-bit mode, fetching instructions two at a time from page-mode flash memory or EPROMs. Yet despite its speed and qualifications, the 4000 fits in just 1 cm^2 (0.16 in^2) and runs from inexpensive crystals.

The 4000's 32-bittedness begins inside, with its register set. As Figure 2 shows, the chip's many registers can be addressed as 8-bit registers (à la the Z80), or concatenated as 16-bit register pairs, or even as six different 32-bit registers (twelve if you count the alternate registers). Having 32-bit

registers helps immensely for addressing memory; the processor can now access any location in physical memory via a 32-bit pointer register. Awkward banking, page-switching, or swapping become just an ugly memory.

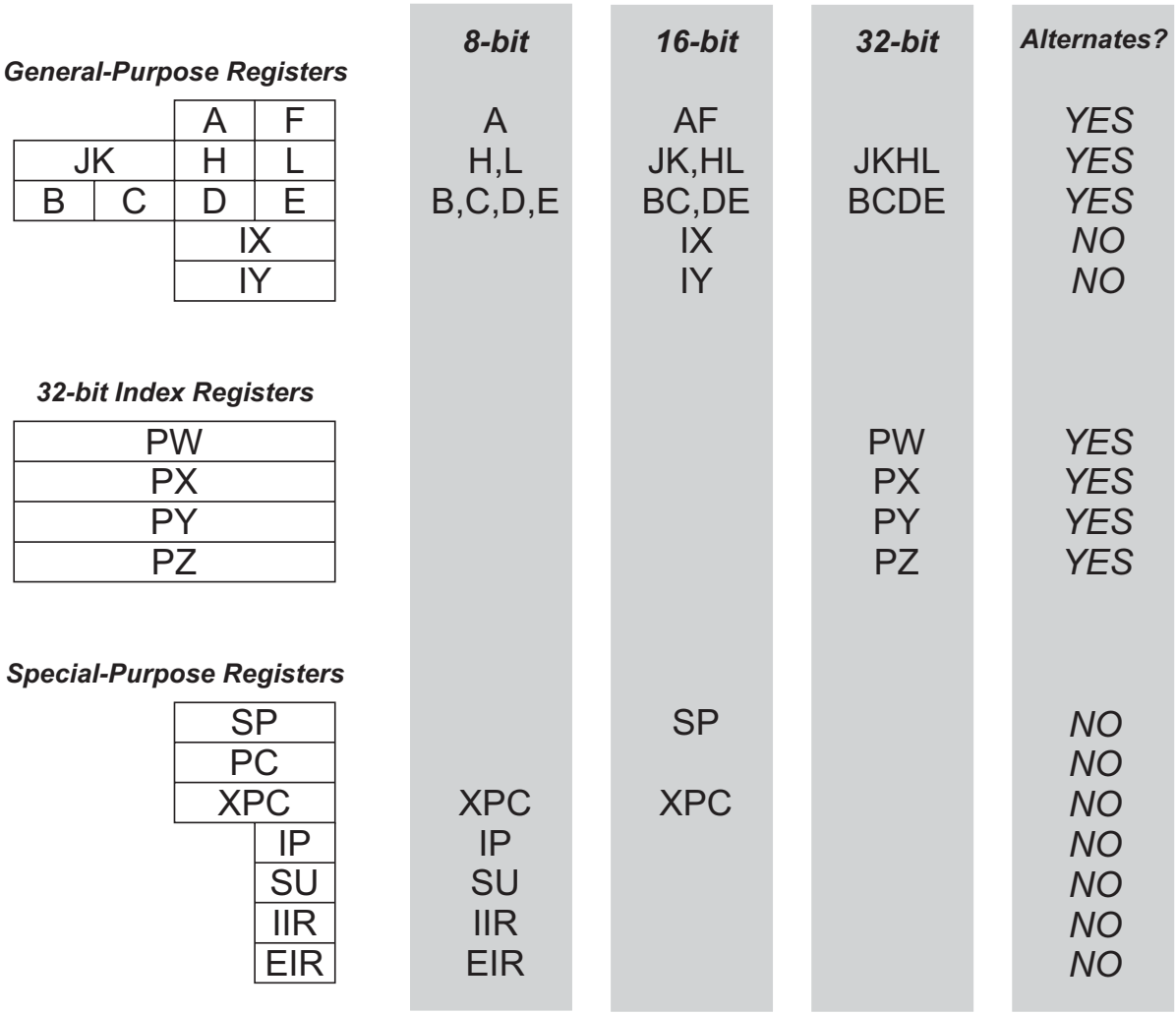


Figure 2. Register Map of the Rabbit 4000

Rotates and shifts are among the cryptography programmer's favorite bag of tricks. On the Rabbit 4000 these are 32-bit operations, without having to concatenate shorter registers or pass everything through an accumulator. Shifts and rotates work in both directions, and by convenient 1-, 2-, 4-, or 8-bit amounts. Logical operations, too, now extend to 32 bits, as do addition and subtraction. By digesting keys and clear text in 32-bit gulps, the 4000 clears its encryption plate in record time.

But wait, there's more. Two complex new table-lookup instructions implement the AES "Sbox," a 256-byte lookup used for every byte of AES-encrypted data. By implementing this complex function (including the data table) entirely in hardware, the Rabbit 4000 saves code space, memory space, and gobs of time.

It gets better. The processor includes 32 bytes of tamper-resistant battery-backed SRAM inside the chip. Like any nonvolatile memory, this RAM can be used for storing variables, settings, or parameters. But because it's tamper-protected, it's ideal for cryptographic keys or hash values. By storing secret keys away from off-chip memories (like external flash chip or ROM), the 4000 can keep the key data away from prying eyes and logic analyzers.

The chip's tamper protection guards against attacks that attempt to learn the contents of the secure SRAM. Any attempt to bootstrap the processor, either through external means or through software, erases the SRAM. Key data stored in the secure SRAM never has to leave the chip.

Clearly the 4000 has the hardware to handle real encryption and decryption, as well as a convenient Ethernet interface for network-enabled products. But what about the software? Encryption algorithms are complicated, and writing new code to adhere to the standard (or standards) doesn't excite most programmers. Fortunately, Rabbit provides all the "middleware" you need to get crypto-enabled systems up and running.

SSL and HTTPS are the two methods used to secure all Web traffic, whether it's credit card information or restricted site access. Rabbit supplies both of these for its 4000 chip, as well as for its recently upgraded Rabbit 3000A processor.

Unleash the Rabbits

Nearly any kind of embedded system can be Web- or Internet-enabled these days. The proverbial Internet Coke machine is already a reality, and for good reason. Bottlers don't want to send a truck out to refill a soda machine that's not empty, nor do they want to ignore an empty machine that needs refilling. A vending machine that can report on its own status or request service saves everyone time and frustration. Municipal gas and water meters are going online so the local utility company doesn't have to employ meter readers to walk up and down every single street. Office equipment is nearly always networked now, and even consumer electronics are sprouting wired or wireless network connections. A network interface is no longer a product; it's a feature in another product.

"Imagine a 'smart connector' that handles security protocols as the data passes through"

Security isn't limited to big boxes, either. Smart cards – which are hugely popular in Europe but still almost unknown in the United States – are always secure. Seemingly nothing more than plastic credit cards, they're actually secure, encrypted storage with a processor. They're a universal pass for bank ATMs, point-of-sale terminals, admission tickets, medical records, rental billing and what have you. Similar cards make handy "keys" for television set-top boxes, satellite receivers, building security, and more.

The Rabbit 4000's small footprint and built-in Ethernet MAC and PHY suit it well for deeply embedded network connections. The chip needs little more than an RJ-45 connector to become a full-fledged Internet node. The final product might be no bigger than the Ethernet jack itself. Imagine

a “smart cable” that converts Ethernet to USB, or that plugs into a common RS-232 port. A “smart connector” could handle security protocols invisibly, as data passed through the interface.

Industrial or robotic systems can make use of the 4000’s two quadrature decoders. Add in the Ethernet connection and you’ve got a remote sensing product for anything from agricultural sprinklers to industrial assembly lines. The 4000 can output as well as input, too. Its PWM (pulse-width modulation) outputs can drive stepper and servo motors, possibly from commands received (securely) over Ethernet. Dozens of input-capture pins, general-purpose I/O pins, and a half-dozen serial ports allow the 4000 to connect to pretty much anything.

Burying the 4000 inside a cable or DSL modem would add security to shared home connections to the Internet. Better still, a router connected to the modem could provide its own security for incoming and outgoing data (even securing local clients from each other), all invisibly and at next to no cost. With size, cost, and power all so small as to be essentially irrelevant, the 4000 can be used literally anywhere an Ethernet jack appears, adding extremely local intelligence – and security – to any Ethernet connection.

Z-World, Inc.

2900 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-3737
Fax: (530) 757-3792

www.zworld.com

Rabbit Semiconductor

2932 Spafford Street
Davis, California 95616-6800
USA

Telephone: (530) 757-8400
Fax: (530) 757-8402

www.rabbitsemiconductor.com