

Using Elliptic Curve Cryptography (ECC) for Enhanced Embedded Security

**Financial Advantages of ECC over
RSA or Diffie-Hellman (DH)**

Jerry Krasner, Ph.D., MBA

November 2004

Embedded Market Forecasters

American Technology International, Inc.

Embedded Market Forecasters
Research and Consulting
for Embedded Products,
Markets and Channels



Executive Summary

Embedded security is the next important product differentiator that can greatly expand markets for embedded vendors, and create the need for new products for embedded developers.

Implementing security protocols in embedded systems is not an easy proposition and systems are often delivered that, for reasons of complexity, limited resources, or implementation, fail to deliver required levels of security. Embedded systems are so frequently limited by memory, microprocessor performance (or required number of gates) and by power drain, that lower levels of security are installed than required to protect the information.

The need for security in embedded applications is easily stated but difficult to apply. There is a real cost to implementing embedded security and there are alternative security methodologies that can be used to enable adequate security while minimizing development cost and product degradation. Comparative costs derive from the requirements of different security implementations for memory, processor speed/gates, power drain, and bandwidth.

Twenty years ago the state-of-the-art in processor technology permitted 33 million instructions per second – today that figure is 4500 times higher. Hence what was considered an unbreakable encryption algorithm in 1984 is now declared useless today. In the realm of symmetric key cryptography, DES' 56 bit key is now history; it does not offer adequate security given the resources available to do brute force attacks – attacks which check every possible key. It is for this reason that the standard, which once defined DES as the appropriate symmetric algorithm to use for the encryption of sensitive information within the U.S. government, has been declared obsolete. Now the Federal Information Processing Standard (FIPS) 197 specifies the recently developed AES algorithm, whose keys can offer 128, 192 or 256 bits of security.

AES is such a powerful encryption algorithm that it is estimated that if it took only one second for a computer system to crack DES, it would comparably take the same machine 150 trillion years to break a 128 bit AES algorithm! Notwithstanding this extremely high level of encryption, AES also offers improved performance.

However, that's only part of the security equation. In order to provide the level of embedded security offered by AES, it is necessary to provide an equal level of asymmetric encryption. Only ECC can provide the required public key security strength with a small key size. In this paper the reader will become familiar with Elliptic Curve Cryptography (ECC) and how it provides higher levels of security while reducing required memory, processing power, power drain and required bandwidth.

The purpose of this paper is actually threefold; to acquaint embedded developers and vendors to the market opportunities that embedded security provides; to explain the need for public/private key cryptography; and to present ECC as a design solution for cost savings and product enhancement.

I. The Emergence of Embedded Security

Overview

Technology guru Michael Murphy describes the three-waves of technology growth for semiconductors. The First Wave occurred in the mid 70's and was driven by mainframe computers; the Second Wave occurred in the mid 80's and was driven by the PC revolution; the Third Wave occurred in the 90's and was driven by the Internet explosion (not to be confused with the dot-com sector). Each Wave generated a 500% growth over the preceding Wave. Murphy, along with embedded market researchers (e.g., Embedded Market Forecasters – EMF), have expressed the view that we are at the onset of the Fourth Wave – Murphy calls it “Universal Connectivity”, EMF calls it “Communications & Connectivity”.

The technology is at hand to provide the bandwidth and software for an “electronic skin” that promises to change the way business is conducted. Manufacturers will be connected to their devices to offer better service; every printer, elevator, air conditioner, vending machine, etc., can report its status, financial receipts and maintenance requirements as they occur. Millions of workers can connect communications devices through their virtual private network (VPN) to support sales, CRM and product availability. This ability will only increase as the world moves from 2.5G to 4G wireless technology.

The key to this emerging technology wave is the availability of a reliable, fail safe, inexpensive and secure connectivity technology for embedded applications.

Unlike their larger, higher-powered cousins that are found in server bays, telecom racks and in industrial automation, embedded devices are largely RAM and middleware limited, memory and processor constrained and power restricted. The security requirements for this huge base of connected devices are unique and must be addressed in a different manner. As you read on you will discover why Elliptic Curve Cryptography is essential for embedded security.

Embedded security is the next product differentiator for embedded devices. The unusual design constraints placed on embedded devices require a new, highly efficient, easy to deploy cryptography scheme that provides high levels of security while minimizing memory, execution speed requirements and power requirements.

ECC is an essential methodology for meeting these requirements of embedded designs.

The key to this emerging technology wave is the availability of a reliable, fail safe, inexpensive and secure connectivity technology for embedded applications.

Embedded Encryption (or lack of)

The confidentiality and integrity of sensitive information is a critical component of any secure system. Typically, this protection is implemented at least in part through the use of symmetric algorithms, like DES, AES, or countless others.

Unfortunately, many networked embedded systems lack robust encryption to protect sensitive information. This may be due to resource limitations (strong encryption requires substantial processing, memory, and power), cost restrictions, design limitations, or possibly the extension of an internal, legacy, or hard-wired system onto an open network such as Ethernet or Intellectual Property, without considering the associated security implications.

Regardless of the reason, the potentially disastrous results are the same. Intruders or malicious insiders can read, intercept, modify, or remove communications at will. If proprietary wireless RF links are involved, the danger is further amplified, as anyone with suitable equipment can attack the system, potentially from a substantial distance given a high-gain antenna.

In many cases, damage resulting from eavesdropping on sensitive information pales in comparison to damage resulting from forged or modified communication. Consider a gas pipeline monitoring system, which uses wireless RF links between sensor nodes along a gas pipeline, which monitor and report on line pressure, temperature, purity, and other critical data. If the system lacks strong security, an attacker could easily damage or destroy the sensors at a vulnerable point on the pipeline, then substitute his own device which generates false sensor data, while the attacker damages the pipeline. Alternatively, the attacker could generate false readings indicative of a leak or fire, diverting maintenance and response personnel from the intended point of attack.

Clearly, insufficient cryptographic protection can lead to substantial compromises, many of which are not immediately obvious at system design time. A prudent embedded system designer must consider the implications of intercepted, deleted, modified, and forged information from all components of a networked system, and take steps to provide encryption to protect against such attacks.

Defining the Marketplace

We are entering an age of unlimited bandwidth and enhanced connectivity. Security will be a necessity of all embedded systems that employ connectivity of any sort. Hackers have not yet focused on embedded systems (traffic lights, power plants, industrial controls) but the chances of such as we move into a new age of connected devices is a virtual certainty.

The service industry will take on new dimensions as connected devices become pervasive. Home appliances that can be remotely monitored or investigated will dramatically lower the cost of service and make such offerings more profitable. Can a manufacturer of refrigerators or microwave ovens afford to send service people out on warranted service plans without knowing what part or component is required when their competitors can service these systems remotely? Neighborhoods could use wireless connectivity to enable the reading of all meters from all homes – provided that absolute security was built into the connectivity once the transmission reached the wireline.

Point of sale systems depend on dial-up connectivity and fault tolerance combined with security. The economics of connectivity again dictate the required technological solution. Credit card processing companies have stated that if they could save one cent per transaction, they would realize an annual savings (direct to the bottom line) of \$70 million.

Embedded systems have grown exponentially over the last decade. They now control every thing from automobile subsystems to complex medical devices. With the advent of WiFi and sophisticated networking, more embedded systems are being used to monitor and report data back in traffic systems, utilities, and medical devices. Embedded devices now control factory automation systems, nuclear power, and telecommunication systems.

Furthermore, there is an emerging services market for remotely accessing home appliances when service is required to report either potential or actual problems so that they can be cost effectively addressed. In addition vendors and utilities are seeking communication methods to lower the cost of remote reading of electric and gas meters.

All of these systems are subject to failure or potential disruption or compromise from some external threat. In the case of failure, according to government reports, the data that is lost costs the economy billions of dollars annually. In the case of purposeful disruption or compromise, the cost of one single event, as we've seen in recent history, can be measured in significant loss of human life and billions of dollars.

Security Considerations for Embedded Applications

Embedded systems are responsible for the availability and functionality of many critical systems, from factory automation to gas pipeline monitors to networking equipment. Unfortunately, the critical importance of embedded systems is seldom matched with a strong, comprehensive security infrastructure.

As a result of the substantial difficulty in the design and implementation of secure protocols, the consensus among the security community has it that system designers are well advised to use existing, proven security protocols, from trusted, commercial vendors, rather than develop their own protocols or implementations.

Building security into embedded devices will require a concerted effort on the part of embedded software developers, OEM's building embedded systems, vendors selling them, and customers purchasing and implementing products. Until information security becomes a strategic technology for embedded systems developers, their products will continue to be characterized by complacency and vulnerability.

II. Public/Private Key Cryptography¹ in Embedded Designs

The complexity of security

The art and science of cryptography is a complex one; so much so that PhD's in abstract mathematics with years of experience in cryptography often design encryption systems that are found to be broken upon peer review. Of the many examples with which broken encryption can be introduced into a product, consider the widely publicized cryptographic vulnerabilities in WEP, the security component of the 802.11(b) Wireless Ethernet standard developed by the IEEE.

An equally dangerous (and less obvious) hazard to the security of a system—particularly an embedded system—is the false sense of security which arises from the use of weak cryptographic techniques, or worse still, of “scrambling” or encoding techniques which claim to provide ample security while actually providing very little protection.

AES: The Right Security Level

The National Institute of Standards and Technology (NIST) has published its forecast for adequate security for the ensuing 30 year period (as presented in the table below). Their recommendations are predicated on the Advanced Encryption Standard (AES) 128 bit symmetric security and their forecast for microprocessor capability to break asymmetric encryption.

minimum bit-security level	80	112	128
protection lifetime of data	present-2010	2011-2035	2036 and beyond

source: NISTSP 800-57

Based on this information, NIST and the National Security Agency (NSA) have standardized on AES and have developed their recommendations and requirements for asymmetric encryption based on the fact that the asymmetric algorithm must match the equivalent security level of the symmetric algorithm.

In fact, NIST notes in FIPS 140-2: Security Requirements for Cryptographic Modules that “Compromising the security of the key establishment method (e.g., compromising the security of the algorithm used for key establishment) shall require at least as many operations as determining the value of the cryptographic key being transported or agreed upon.” This means that some cases of weakness are not a result of an incorrect application of strong encryption, but rather from using weaker and stronger cryptography together.

¹ A discussion of how Public/Private key cryptography works is presented in the appendix.

Symmetric and asymmetric cryptosystems

By way of analogy, consider a chain composed of several extremely strong links, and one very weak one; despite the strength of most of the chain, the single weak link compromises the security of the entire chain. The strong-as-the-weakest-link axiom is equally applicable to system security, with the notable difference that weak links are very easy to introduce by accident, and very difficult to detect. Therefore, all links must be equally secure, or the overall security of the system is less than the security of the individual links.

As an example, SSL provides a virtual “tunnel” across a network, which can be used to send information protected from interception and modification. This is accomplished by using two forms of cryptosystems together: asymmetric and symmetric. The asymmetric cryptosystem is used to negotiate a common key between the client and the server for use during their communication, and the symmetric cryptosystem is used to encrypt and decrypt traffic with the negotiated key. Common asymmetric algorithms include RSA and Diffie-Hellman (DH) while typical symmetric algorithms are DES and AES.

To continue the analogy from above, the asymmetric system is one link in the security chain, and the symmetric system is another. If the asymmetric encryption is broken, an attacker can determine the negotiated common key, and access the traffic using that key, without having to break the symmetric encryption. Similarly, if the attacker can break the symmetric encryption, he can access the traffic without breaking the asymmetric encryption. Given this scenario, clearly the symmetric and asymmetric schemes should be of equal strength.

Unfortunately, as stronger symmetric algorithms like AES have come into common use, the corresponding asymmetric encryption has not increased in strength to match. As a result, many systems use 128- or 256-bit AES for symmetric encryption, yet rely on 1024-bit asymmetric encryption. Thus, these systems have a security level roughly equivalent to a system using 80-bit keys for symmetric encryption.

While 80 bits is still a substantial level of security, it is not the 128- or 256-bit level that is likely advertised, and the systems are incurring the additional cost of 128- or 256-bit keys, without any additional security, which is wasteful of potentially limited resources. This is exactly the situation that NIST was trying to solve in FIPS 140-2: Security Requirements for Cryptographic Modules when mandating that the asymmetric algorithm match the symmetric algorithm in terms of security strength.

And though SSL was used in this example due to its near-ubiquity as the protocol of secure web transactions, the key size disparity is an issue anywhere asymmetric and symmetric cryptosystems are used together.

Advantages of ECC for design efficiencies

So now let's look at what level of public-key cryptography would be required to match the strength of AES.

A typical 1,024-bit RSA asymmetric key is about as secure as an 80-bit symmetric key, yet AES key sizes range from 128 bits to 256 bits. To provide security equivalent to AES, RSA public-key sizes would have to range between 3,072 and 15,000 bits long, which is so big that typical embedded hardware would be unable to maintain reasonable levels of performance or throughput.

One appealing solution to the key size disparity problem is the promising family of asymmetric algorithms known as Elliptic Curve Cryptography, or ECC. ECC uses much smaller key sizes than other asymmetric techniques, while providing equally strong security. Therefore, while a 128-bit symmetric key would require an RSA or DH key of 3,072 bits in order to provide equal protection. An ECC key of 256 bits would provide just as much security, at less than 1/12th the size. The benefits are more substantial for larger key sizes: a 256-bit symmetric key should be protected by a 15,000-bit RSA or DH asymmetric key, while an equivalent ECC asymmetric key size is only 512 bits.

Due to the difficulty in breaking its encryption, Elliptic Curve Cryptography can provide the same level of RSA or DH encryption at a greatly reduced bit size. This is very important to embedded developers and vendors for whom power drain, memory, processor requirements and bandwidth requirements are limited and of concern. There is an obvious cost savings to the use of ECC—a topic that we will address in the ensuing sections.

The following table shows the comparative ECC and RSA bit size requirements for five different symmetric algorithms in order to achieve different bit-security levels.

Symmetric key algorithm	Skipjack	3-DES	AES-128 small	AES-192 medium	AES-256 large
Hash algorithm	SHA-1	SHA-256	SHA-256	SHA-384	SHA-512
Bit-security level	80	112	128	192	256
ECC size (prime)	192	224	256	384	512
ECC size (binary)	163	239	283	409	571
RSA modulus size	1024	2048	3072	7680	15 360

sources: FIPS 186-2, NIST SP 800-57, ANSI X9.30.1 - 2002

ECC size	192	224	256	384	512
RSA modulus size	1536	4096	6000	>10 000	>20 000

source: NESSIE Security Report (2003)

Key Size Comparisons

ECC offers considerably greater security for a given key size. That smaller key size also makes possible much more compact implementations for a given level of security, which means faster cryptographic operations, running on smaller chips or more compact software. This means less heat production and less power consumption – all of which is of particular advantage in constrained devices, but of some advantage anywhere. There are also extremely efficient, compact hardware implementations available for ECC exponentiation operations, offering potential reductions in implementation footprint even beyond those due to the smaller key length alone.

Since ECC is an appealing solution to the key size disparity problems, embedded implementations of ECC are now being designed into systems. While several standard security protocol implementations do support ECC, RSA is more widely deployed. This will surely change as ECC gains in popularity following the standardization by the NSA.

III. Financial Advantages of Using ECC over RSA

Embedded systems generally are limited by their available resources: battery power, memory, bandwidth, and processing capacity. For many embedded designs that require adequate levels of security, the additional memory required and the CPU time taken make the use of RSA and DH impractical – if not impossible – for implementation. In other embedded designs, the additional overhead and associated cost required to implement DH or RSA is excessive.

Because security is a support function for the real intent of any device, it should be as unobtrusive and undemanding on the device as possible. As the previous section illustrated, today's de-facto standard, RSA, requires increasingly large key-lengths in order to provide acceptable security.

Continuing to use RSA means that an ever-increasing portion of a device's processor must be used for security operations. This is why ECC is a compelling alternative to RSA or DH for embedded designs. Using ECC means that an embedded device can:

- use a smaller, cheaper processor or apply more processing to the main functions of the device;
- in cases where the security operations are integrated into the chipset, provide gate counts for ECC that are significantly smaller, meaning less real-estate and lower chip costs;
- apply fewer processor cycles because the device is creating less heat and therefore less power drain – meaning that battery life is longer;
- require less bandwidth for transactions due to more-efficient protocols

Associated Design Costs for Embedded Systems

This section examines the associated cost factors that embedded developers, OEMs and vendors should consider when implementing security into their embedded designs.

THE MATH BEHIND ECC PERFORMANCE

Let's now turn for a moment to consider the differential cost of implementing DH versus ECDH (EC-Diffie-Hellman).

A Diffie-Hellman key agreement consists of two exponentiations by each party. Often the exponents are taken to be no longer than 160 bits since this does not appear to lessen the security. For DH, one makes a calculation by computing $g^e \text{ mod } p$, where e is 160 bits long, consists of 160 modular squarings and 80 (on average) modular multiplications.

In the elliptic curve case, the DH key agreement consists of two point multiplications by each party. For the same security level as the DH of length 1024 bits, we require a curve whose prime order subgroup is around 160 bits long. The upshot of this is that the elliptic curve computations will be done with 160 bit arithmetic. Now, computing eG , where e is a 160 bit integer, and G is a point on the elliptic curve, we require 160 point doubles, and 80 (on average) point additions.

Therefore, to see how much faster an Elliptic Curve Diffie-Hellman is compared to a Diffie-Hellman, we compare how long a point double/addition takes compared to a modular squaring/multiplication.

For an elliptic curve E over a field $F\{2^m\}$, the dominant costs in a point double are two field multiplications and one field inversion. The cost of a point-add is roughly the same. On limited power devices, a field inversion costs about 2.5 field-multiplies. (On machines with caches and pipelining inversion can cost more, up to 4.5 or more field multiplies). So for a rough estimate let us say that a point operation costs us five 160 bit field-multiplies.

Since the complexity of multiplication is quadratic, (for the sizes we are talking about), doubling the number of bits means quadrupling the time for a multiplication. Thus, a 1024 bit multiply takes roughly $(1024/160)^2 = 40$ times as long as a 160 bit multiply. Similarly, a 3076 bit multiply equals 144 times as long as a 256 bit multiply.

continues...

Processing Power

In determining the potential operations of any device, chip selection is one of the main considerations: A more expensive processor allows a device to do more. Using more efficient methods allows a less expensive processor to perform the same functions. This is why ECC is so critical to embedded devices, where processing, size and functionality provide clear trade-offs.

The sidebar describes the mathematics of why ECC is computationally more efficient.

As an example, a smart card, connectionless key fob or other small device that is used for a financial transaction must authenticate itself to the reader. Performing this transaction using ECC takes significantly less processing, meaning that two benefits are possible:

1. The transaction will happen more quickly, meaning that the system can process more transactions and generate more revenue; or
2. A less expensive (and slower) processor can be used in the smart card to perform the same transaction in the same amount of time.

Mobile devices such as PDAs and smartphones show similar results. A typical processor for these types of devices is the ARM SA1110, rated at 206 MHz (tested in an iPAQ). At the 128-bit-AES-equivalent security level, ECC-256 provides excellent response times; RSA-3072 offers good response only in signature verification; the times for signature and key generation are unacceptable. The delay is prohibitive for many – if not most – embedded applications.

Algorithm Security Strength 128 bit	Crypto Function	Response times
RSA – 3072	Decrypt/Sign Verify/Encrypt	670 ms 18 ms
ECDSA/ECDH – 256p	Sign Verify	7 ms 18 ms

Therefore, to offer equivalent security and performance on a device using RSA-based security, you would need to use a faster processor which would increase the costs of the embedded device.

Since a point operation requires 5 field multiplies, we expect that a 1024 bit multiply will cost $40/5 = 8$ times as much as a 160 bit multiply. Therefore if modular squarings cost the same as modular multiplies we can expect the Elliptic Curve Diffie-Hellman to be 8 times faster than Diffie-Hellman, since the basic operations are 8 times faster and we are doing the same number of basic operations. Sometimes squarings can be a little faster than modular multiplies, but not more than twice as fast, and in this case the EC Diffie-Hellman is still 5 times faster.

We can summarize with a “rule of thumb” that ECDH, mathematically speaking, is 5-10 times faster than DH and 5-10 times smaller in terms of bandwidth. In practice, as we will see ECC-based algorithms can offer performance improvement that are substantially higher.

Gate count

As shown, ECC offers improvements in software. However, it can be particularly efficient in hardware. As computing environments move to trusted environments and hardware-based implementations of security functions, the benefits of ECC will be increasingly dramatic in comparison to RSA and DSA. Optimized chip designs have been shown to be as much as 37 times faster than comparable implementations in software.

This advantage will be seen in both gate counts (which indicate the space taken up on a chip) and performance in comparison to RSA. As all chip designers know, more gates cost more money. They are also important as system designers move to put more functions onto a single chip and to build fully-integrated system-on-a-chip hardware.

As can be seen in the following table, ECC stands out in both performance and hardware real-estate. Following normal technology trends, the newer technology is smaller by a factor of ten (3,260 gates against 34,000 gates) when optimized for space, and still shows performance several times better than its older rival. When optimized for speed, ECC is more than 7 times faster at current key lengths (2.6 ms against 0.35 ms), and more than 80 times faster when using key lengths that will be required for future-proof security.

Algorithm	Optimization	Time	Gate count
RSA – 1024 ECC – 163	Space- optimized	4.90 ms 0.66 ms	34,000 3,260
RSA – 1024 ECC – 163	Speed- optimized	2.60 ms 0.35 ms	150,000 48,400
RSA – 3072 ECC – 283	Space- optimized	184 ms 29 ms	50,000 6,660
RSA – 3072 ECC – 283	Speed- optimized	110 ms 1.3 ms	189,200 80,100

For these reasons, ECC is a clear choice, and an obvious one when implemented in hardware. Any device that is using RSA is making a significant sacrifice in processing and giving up more microprocessor real estate than it should.

Battery drain

Because a more efficient implementation requires less processor cycles and less work, less power is used and less heat generated. This is critical to mobile devices, where the limiting factor to usage in many cases is the battery life. In addition, the potential for embedded devices improves as concerns over heat dispersion are alleviated.

Therefore ECC can be seen as part of an overall slimming down of devices, another innovation in the long running miniaturization of electronics. In comparison to the vacuum tube, the transistor offered smaller size, less power drain and better operational speed; ECC is to RSA as the transistor was to the vacuum tube.

Bandwidth and protocols

Using more efficient key agreement methods such as ECMQV further streamlines security in embedded devices. Bandwidth, which can be at a premium in embedded environments such as mesh networks, RFID, or specialized networks like air-traffic control, can be used more efficiently using these protocols. More broadly, any network can be made more efficient by moving to ECMQV from Diffie-Hellman or another key-agreement method.

For embedded systems, ECMQV has two major benefits beyond its security:

- The computational requirements are non-intensive, meaning that processing (and therefore power consumption) is kept to a minimum
- Communication overhead is low: Each side sends only one elliptic-curve point to the other. For ECC-163 (the equivalent of RSA-1024), this is 326 bits, or 41 bytes of payload – effectively two tiny data packets, acceptable to all but the most bandwidth-challenged systems. In fact, the 326 bits can be reduced to 164 if point compression is used.

Using less bandwidth can also affect transaction time and capacity, meaning that more can be done in the same amount of time. A voice-over-IP (VoIP) system, in which a softswitch may handle millions of signaling connections for call setup on a daily basis, and over which the calling parties may then make secure connections, could clearly benefit from less intensive security protocols.

Implementing these benefits together as an embedded system builds real value. The inherent computational advantages of ECC, plus its affinity for hardware, make the design choice easy.

Real Benefits for the Real World

Throughout this paper we have focused on embedded systems and the benefits of using ECC for security. However, we have also alluded to the fact that the reason this security is required is that these embedded systems are also connected and exchanging information. This section will show how ECC has significant benefits for both servers and clients.

Server-side benefits

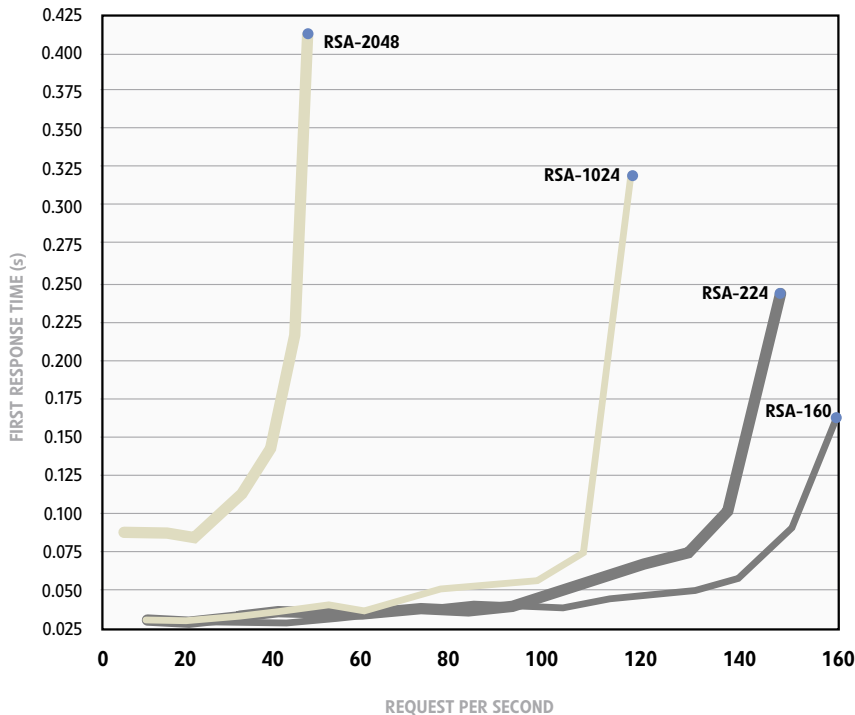
Using ECC in lieu of RSA, servers are enabled to do more with less – crypto takes less time, less processing; and a server can handle more connections/transactions. RSA’s defense is to throw more hardware (and expense) at the problem; ECC instead offers more capacity from the same server hardware.

Examples of servers for which ECC can offer improved server performance:

IPSec servers: Secure connections can occur faster, and the number of secure connections can be larger because the keys are smaller.

- Voice-over-IP gateways which handle secure connections or signaling.
- SSL servers benefit, whether serving connections to clients using mutual authentication, or one-way authentication only.

A Sun Microsystems study shows clearly the benefits of using ECC on SSL servers. At equivalent security levels (RSA-2048/ECC-224), a web server can complete 3.5 times as many secure connections using ECC as a server using RSA.



Client-side benefits

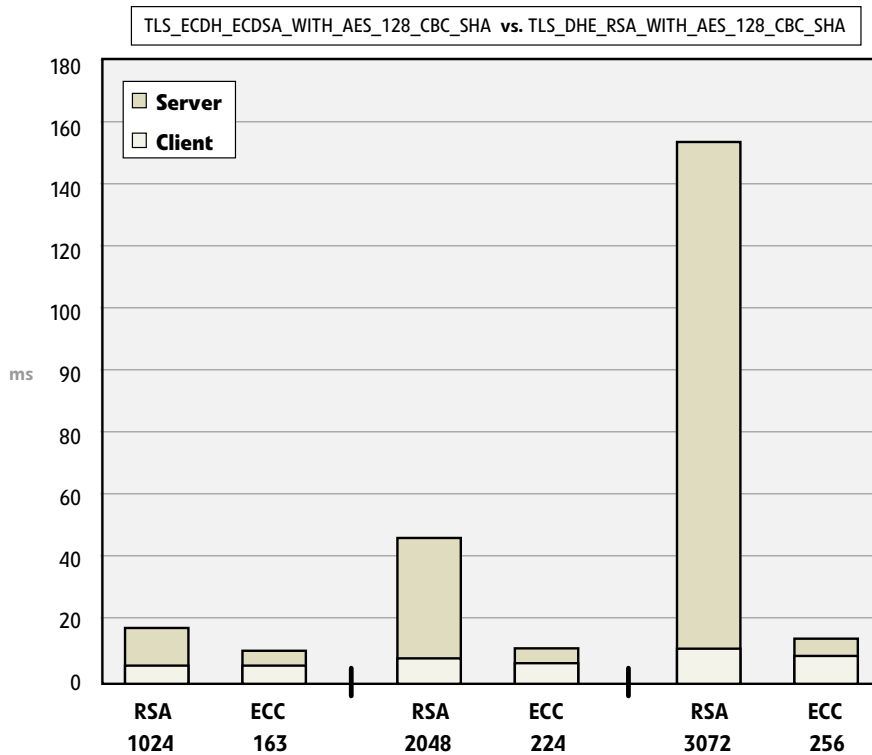
ECC enables a larger range of clients to access a server; for constrained clients, it is the only answer; for larger clients, performance is improved.

The benefits of ECC in a client are most impressive when a protocol calls for client authentication (or mutual authentication). This occurs during most modern-day interaction: Identification of a user or device, and most transactions. Ordinarily, this takes the form of each side issuing a challenge to the other; the challenges are digitally signed and returned to the other party for verification.

Verification times are fast: Both RSA and ECC return similar values, in most cases only a few milliseconds. RSA is faster for present-day key lengths; ECC becomes faster as key lengths grow past RSA-1024/ECC-163.

Signing of a challenge, however, shows a factor-of-ten differential in favor of ECC, even at RSA-1024. At RSA-2048 and beyond, generating the signature required for authentication becomes a cumbersome operation.

In the chart below, total transaction times for TLS (SSL) connections are compared for ECC and RSA at several security strength levels. As can be seen, at today's key lengths, ECC offers marginally-better improvement. At key-lengths that are being recommended today (RSA-2048/ECC-224), the performance disparity becomes wide. At future security grades, transactions performed using RSA become unusable (and therefore there are no figures available), while ECC still offers good response times.



Platform: Pentium III 930Mhz (Linux)

Designing for the future

The previous section showed the clear benefits of using ECC technology rather than legacy RSA technology. And in the real world, these technology benefits offer tangible results – reduced expenses and increased revenues.

Given the broad range of potential technology solutions manufacturers and developers may adopt and the variation in business requirements, it is impractical here to detail individual ROI scenarios. Each system developer will do this based on the nature of its business, the stage of evolution of its product or service portfolio, and its objectives in implementing security. The two main factors in product design are how to keep expenses down while providing a saleable product.

Reduced expenses

There are cost savings at each step along the way. As we have seen, servers and gateways that handle many connections can be implemented more cheaply per connection. For both embedded clients and high-performance servers, ECC in nearly all cases offers improvements in speed and memory requirements.

And reducing expenses can mean strong returns. The use of smart cards is one example where ECC offers clear benefits. One of the main functions and major processor uses of a smart card is cryptography; smart cards are generally in use because they can be used for secure access, financial transactions, identification; all of which are security-dominated activities. As RSA key-lengths move beyond 1024 bits, a math coprocessor, meaning more complexity and more expense, becomes essential to provide acceptable performance. Using ECC, however, no coprocessor is required; or, using a much smaller set of hardware to handle ECC computations, smart card transactions happen much faster.

One of the main inhibitors of smart card deployment has been expense; as compared with magnetic-stripe cards, microprocessor chip cards are significantly more expensive. Magnetic stripe cards cost between \$.10-.50/card; microprocessor smart cards cost \$1-3. Deployments are taking place, but would certainly be accelerated by making smart cards less expensive. Because ECC takes less chip real estate and less memory and can remove the need for a co-processor, lower microprocessor costs can be gained by using ECC instead of RSA. The use of ECC could result in increased use of smart-cards and foster improved services for consumers.

Revenue opportunities

There is additional revenue potential, too. ECC enables low bit rate and low power devices to run securely; it enables systems to be implemented in low bandwidth situations. And it enables services to be offered to constrained clients – small wireless devices, tiny sensors, clients with limited battery or processing power. Using RSA, these systems are not possible.

One example is ZigBee, a developing standard for low bit rate, low power wireless devices for consumer electronics, home and building automation, industrial control, PC peripherals, medical sensors, toys and games. System requirements are:

1. Provide 20 - 250 kbps connectivity,
2. Have very low power requirements,
3. Be less expensive than Bluetooth,
4. Provide a secure channel,
5. Small code size: 4KB - 30KB of RAM/ROM, versus over 250KB for Bluetooth

ZigBee chips are expected to quickly fall below \$5/chip and as low as \$1-2/chip. This leaves no room for adding costly chip real estate or memory; the security must be as efficient as possible. ECC is the natural choice: Modern cryptography to meet the needs of a modern system design. It is the only choice to establish a secure channel within a system that requires a small footprint, low power usage, and low bandwidth. Additionally, implementing ECC in a small hardware module as part of an inexpensive ZigBee chip more clearly highlights the benefits.

Other factors

Beyond producing a useful product to generate revenues while maintaining expenses at a minimum, other critical factors include:

- Compliance with industry standards: RSA is aging as a technology, and ECC is being introduced into the major new standards initiatives.
- Government security requirements: The U.S. National Security Agency (NSA) has licensed ECC for use in “critical security infrastructure”; it is the asymmetric algorithm that will be accepted for future government use. If a product is being designed with the possibility of sale to the U.S. government, it must be secured using ECC.
- Risk mitigation: ECC offers a clear upgrade path from RSA. As processing power increases in devices, encryption key lengths will grow to provide adequate security. By designing ECC into products now, a re-design will not be necessary for future versions.

IV. Summary

Embedded Market Forecasters began publishing and speaking publicly regarding its studies on embedded security some 18 months ago – heralding embedded security as the next differentiator for embedded applications. Because of the requirement of the U.S. Government - and its relationship with NIST – for FIPS 140-2 compliance for any communicating technology sold to the US government, an opportunity arose for embedded OEMs and vendors to grab a piece of this lucrative market.

However, because many embedded devices are power, memory and CPU limited, it is clear that there is a need for security implementations that are effective and are able to be implemented while minimizing power drain, incremental memory requirement, and necessary CPU cycles. Elliptic Curve Cryptography provides a compelling advantage in this regard over RSA implementations that were originally intended for other applications.

As well, last year, the NSA adopted ECC for protecting mission critical information of the U.S. government. This decision by the world’s leading codemakers and codebreakers not only validated the strength of ECC, it created the opportunity for ECC to be widely adopted. It seems very likely that other government agencies and the commercial sector will follow suit and adopt ECC for strong security across a wide variety of applications and devices. Taking advantage of this opportunity for significant new revenue streams should be enough of a financial incentive.

However, many developers point to the proliferation of RSA technology throughout the Internet as the reason to stick with this legacy algorithm. While this made sense even 5 years ago, a number of changes have happened in the security industry that point to ECC as the next generation of public-key technology that will replace RSA.

The reason that RSA’s time has come and gone is that increases in processing power and advances in cryptanalytic attacks mean legacy key sizes (1024-bit RSA) that provide only 80-bits of security are just too weak for most industries. The simple solution would be to increase the key size to 3072-bit RSA to match the 128-bits of security provided by AES. However, as demonstrated in section 3 this comes at a cost of 10x performance hit which only increases as we scale the key sizes further. This is obviously a major issue with embedded designs.

It is now just a question of time for the technology to be adopted just as the transition from DES to AES we witnessed recently. Developers should get ahead of the curve and build for the future as there is no doubt that RSA keys will continue to grow and will become too large to meet system needs. Each differential level increases performance and gate requirements, and the cost of implementing RSA versus ECC continues to favor ECC disproportionately. In some cases, the larger RSA key sizes just do not work which is why applications from Postal and Financial to Consumer Electronics have adopted ECC as their public-key technique.

In other applications, there is strong rationale for adding ECC as an alternative to RSA until ECC's adoption is more widespread. This is a practical option as the cost to implement both is quite small. Using the form of ECC mathematics based on prime numbers, like RSA, a cryptographic processor can use the same logic circuits for both algorithms. When this is done, the additional microprocessor gate count required to embed ECC in conjunction with RSA is approximately 10%.

A more efficient version that uses a different style of computational mathematics offers far better performance improvement for systems that require the fastest or smallest security, and can start fresh, without using RSA to interface to legacy security systems. This is the path to design for the future and one that needs to be carefully considered by embedded developers.

Twenty years ago, ECC was still a new cryptosystem and researchers did not know if ECC schemes could be implemented efficiently and securely. Since then, researchers have studied ECC and determined it is a stronger, more efficient technology that is ideally suited for resource constrained environments such as smart cards, cell phones, and personal digital assistants (PDAs). Moreover, ECC systems are also well suited for applications that need long-term security requirements.

This paper has shown how ECC becomes particularly attractive when implementing the stronger symmetric encryption offered by AES. These savings are even more advantageous when computational power, bandwidth, or storage space are limited as is the case in the ever increasing number of embedded systems that are being used everyday. ECC delivers the highest strength-per-bit of any public-key cryptography system known today. Ultimately, the benefits of ECC are many: linear scalability, a small software footprint, low hardware implementation costs, low bandwidth requirements, high device performance.

APPENDIX

Review of Public/Private Key Encryption

Using Public Key Cryptography for Authentication

Diffie-Hellman Encryption

In 1976 the Diffie-Hellman (DH) key agreement protocol (also called exponential key agreement) was developed by Diffie and Hellman and published in the ground breaking paper “New Directions in Cryptography.” The protocol allows two users to exchange a secret key over an insecure medium without any prior secrets.

The problem with the original DH implementation was that a man-in-the middle could intercept the communications between two parties and pose as one of them or they could disrupt the communications. In short, there was no effective method to authenticate the parties to the exchange.

Today, public key encryption using the original Diffie-Hellman protocol has been understood to be an example of a much more general cryptographic technique, the common element being the derivation of a shared secret value (that is, key) from one party’s public key and another party’s private key. The parties’ key pairs may be generated anew at each run of the protocol, as in the original Diffie-Hellman protocol. The public keys may be certified, so that the parties can be authenticated.

Public Key Cryptography

Public key cryptography is a technique that uses a pair of keys for encryption and decryption and for signature generation and verification. Each pair of keys consists of a public key and a private key. The public key is made public by distributing it widely. The private key is never distributed; it is always kept secret.

There are mathematical relationships between the public and the private key that make all of this possible, the most important features of the pair are these: (1) you cannot derive the public key from the private, (2) but you can, nonetheless with appropriate use of algebra, use the public key to verify knowledge of the private key.

In public key signature schemes, these two properties are used as follows to verify identity: the holder of a public key signs a message using the private key and the signature generation algorithm.

The result—the message signature—can then be transmitted to anyone wishing to verify the identity of the person sending the message. If, and only if, the message was genuinely generated by someone with knowledge of the appropriate private key certain relationships between the public key, the message and the signature can be verified—using a signature verification algorithm.

This feature relies ultimately on this unique relationship between the public and private keys. Note again: the scheme allows someone with knowledge of the public key to verify knowledge of the private key, but does not reveal the private key itself. So you can verify someone’s identity without being able to impersonate them yourself—the essential and highly useful feature of asymmetric cryptography.

Suppose Anne wants to authenticate Ben. Ben has a pair of keys, one public and one private. Ben discloses to Anne his public key. Anne then generates a random message and sends it to Ben:

A->B	Random-message
------	----------------

Ben uses his private key to sign the message and returns the signature he generates to Anne:

B->A	{random-message}Bens-private-key
------	----------------------------------

Anne receives this signature and verifies it against her message by using Ben’s previously published public key. This verifies she’s talking to Ben. An imposter presumably wouldn’t know Ben’s private key and would therefore be unable to properly sign the random message.

Handing Out Public Keys

How does Ben hand out his public key in a trustworthy way? Let’s say the authentication protocol looks like this:

A->B	Hello
B->A	Hi, I’m Ben, Bens-public-key
A->B	prove it
B->A	Anne, This Is Ben
	{ digest[Anne, This Is Ben] } Bens-private-key

With this protocol, anybody can be Ben. All you need is a public and private key. You lie to Anne and say you are Ben, and then you provide your public key instead of Ben’s. Then you prove it by encrypting something with the private key you have, and Anne can’t tell you’re not Ben.

To solve this problem, the standards community has developed an object called a certificate. A

certificate has the following content:

- The certificate issuer's name
- The entity for whom the certificate is being issued (aka the subject)
- The public key of the subject
- Some time stamps

The certificate is signed using the certificate issuer's private key. Everybody knows the certificate issuer's public key (that is, the certificate issuer has a certificate, and so on...). Certificates are a standard way of binding a public key to a name.

By using this certificate technology, everybody can examine Ben's certificate to see whether it's been forged. Assuming that Ben keeps tight control of his private key and that it really is Ben who gets the certificate, then all is well. Here is the amended protocol:

A->B	Hello
B->A	Hi, I'm Ben, Bens-certificate
A->B	prove it
B->A	Anne, This Is Ben { digest[Anne, This Is Ben] } Bens-private-key

Now when Anne receives Ben's first message, she can examine the certificate, check the signature (as above, using a digest and public key decryption), and then check the subject (that is, Ben's name) and see that it is indeed Ben. She can then trust that the public key is Ben's public key and request Ben to prove his identity. Ben goes through the same process as before, making a message digest of his design and then responding to Anne with a signed version of it. Anne can verify Ben's message digest by using the public key taken from the certificate and checking the result.

A bad guy - let's call him Mike (in the middle) - can do the following:

A->M	Hello
M->A	Hi, I'm Ben, Bens-certificate
A->M	prove it
M->A	????

But Mike can't satisfy Anne in the final message. Mike doesn't have Ben's private key, so he can't construct a message that Anne will believe came from Ben.

Exchanging A Secret

Once Anne has authenticated Ben, she can do another thing - she can send Ben a message that only Ben can decode:



The only way to find the secret is by decrypting the above message with Ben's private key. Exchanging a secret is another powerful way of using public key cryptography. Even if the communication between Anne and Ben is being observed, nobody but Ben can get the secret.

This technique strengthens Internet security by using the secret as another key, but this time it's a key to a symmetric cryptographic algorithm (such as DES, RC4, or IDEA). Anne knows the secret because she generated it before sending it to Ben. Ben knows the secret because Ben has the private key and can decrypt Anne's message. Because they both know the secret, they can both initialize a symmetric cipher algorithm and then start sending messages encrypted with it.

How secret-key is computed is up to the protocol being defined, but it could simply be a copy of secret.

Authentication with Asymmetric Cryptography

In the case of asymmetric authentication methods – the core technology behind digital signatures and certificates – we normally speak of a private key (in the possession of the entity wishing to prove its identity) and the public key (in the possession of anyone who wishes to verify the identity of the entity possessing the private key).

You may, with the public key, verify that an entity has knowledge of the private key – but you cannot derive the private key from the public. This is the critical feature of asymmetric cryptographic schemes that makes them so useful.

This property is useful for a number of things: it greatly simplifies key exchange, as one example, and it solves one critical problem symmetric cryptography cannot solve – the problem of guaranteeing unique authentication and non-repudiation.

I'm not sure why hashing is described as "symmetric". A hash function has no keys. Hashing and authentication methods – ones for which there is only one key, and both parties in the exchange use it both for authentication and for signature generation – have the distinct disadvantage that they do not, on their own, offer any way to distinguish which party to the exchange signed a given message. If both or all parties must know the key, based on cryptography alone, you cannot distinguish which signed any given message. Any of them could have. In asymmetric authentication schemes, only one party knows the private key, with which the message is signed. Any number may know the public key. Since the private key cannot be derived from the public, the signature serves as a unique identifier. If the message verifies as having been signed by the person with knowledge of the private key, we can narrow down who sent the message to one.

About Embedded Market Forecasters

The premier market intelligence and advisory firm in the embedded technology industry, Embedded Market Forecasters (EMF) is the embedded market research division of American Technology International, Inc. We specialize in providing high-quality data and expert analysis to support our clients' ability to assess the opportunities, risks, and competitive issues involved with developing and deploying embedded technologies. EMF has extensive experience providing both multi-client and custom research on topics including embedded boards, buses, software, hardware and development tools markets as well as embedded technology applications including embedded systems, digital signal processors (DSPs), FPGAs, single board computers, communications/IT, and multimedia. Our clients range from startups to Global 100 companies worldwide. Founded by Dr. Jerry Krasner, a recognized authority on embedded markets, product development and channel distribution, EMF is headquartered in Framingham, Mass.

The Certicom *Catch the Curve* White Paper Series

This paper is the third white paper in the Certicom Catch the Curve White Paper Series. Read the series to find out why the NSA, Research in Motion, Motorola and other leading organizations have adopted ECC.

The series in detail:

The Certicom Catch the Curve white paper series includes three white papers detailing various areas of ECC.

- White paper 1 provides the foundation for understanding ECC, its strengths and advantages
- White paper 2 provides real-world examples of ECC applications, discussing how organizations are using, and benefiting from ECC today.
- This final white paper includes an analysis on the financial advantages of ECC over RSA or Diffie-Hellman. It is written by an independent analyst from Embedded Market Forecasters.

To sign up for all the white papers in the series, please go to :

www.certicom.com/catchthecurve/emf