WHITE PAPER

Industrial
IP Xiphera root-of-trust that works across all Intel FPGA devices

FPGA-Based Security Solutions

Authors

Matti Tommiska
Xiphera Ltd.
Espoo, Finland
matti.tommiska@xiphera.com

Mark Jervis
Intel® Corporation
Programmable Solutions Group
High Wycombe, UK
mark.jervis@intel.com

Karl Wachswender
Intel® Corporation
Programmable Solutions Group
High Wycombe, UK
karl.wachswender@intel.com

Table of Contents

Introduction

The term Industry 4.0 is used to describe the fourth Industrial Revolution[1], where the technology fusion between physical, digital, and biological domains creates interconnected *cyber-physical* systems, including the Internet of Things (IoT). The real-time communication between cyber-physical systems and humans will create new business opportunities, including smart factories, cloud-based automation, and, for example, using artificial intelligence in predictive maintenance.

The long-term promise of Industry 4.0 and IoT is unquestionable, but key technical requirements must be met before its full potential can be reached. These requirements include ubiquitous connectivity, 24/7 availability, low latency, and most importantly comprehensive end-to-end security, all the way from "Sensors to the Cloud". The most important security requirements include protection[2] of IoT devices and the data they produce, process, and transmit, complying with relevant standard requirements, copy protection as well as supply chain security, while at the same time achieving performance requirements, seamless availability, and ease of use.

A major challenge for corporations — and industrial companies are no exception here — is to translate the high-level security requirements listed above into concrete and secure implementations in their own products. This implementation challenge is compounded with historically diverse hardware architectures, very few standards targeted for industrial security requirements[3], overlap and confusion between security and (functional) safety, and in some cases a general lack of knowledge and terminology about security.

Examples of recent highly sophisticated industrial cyberattacks include Stuxnet targeting SCADA systems, Triton targeting industrial control systems, and Crashoverride targeting electric transmission and distribution networks. While not all details of these attacks — and they are just the tip of the iceberg — are known in public, a common denominator is their software-based nature. It should also be noted, that based on publicly available information these attacks did not break the underlying cryptographic algorithms (See also Section 1) — in other words, no *cryptanalytical* breakthroughs were achieved — but rather exploited the weaknesses in the implementation of security.

Consequently, the security challenges of the current Internet[4] have almost always been both created and also solved with software, and there exists a multibillion-dollar industry of companies offering information security solutions to address these challenges (intrusion detection, malware prevention, resilience against (Distributed) Denial of Service attacks, etc.). However, the current software-centric approach to addressing security challenges is not directly applicable for Industry 4.0 and IoT for a number of reasons, including the required lifetime, updateability, power consumption, end product form factor, etc. The advantages of hardware-based security are further discussed in Section 2.

Notes:

1. The previous three industrial revolutions are generally defined to describe the widespread adoption of mechanization and steam power, electricity and mass production, and computers and automation.
2. In this context protection means authentication, confidentiality, and authenticity of data. These are further explained in Section 1.
3. The recently published IEC 62443-4-2 provides the cybersecurity technical requirements for components that make up an IACS (Industrial Automation and Control System), but "component" in the context of IEC 62443-4-2 does not refer to semiconductor components.
4. Sometimes also called the "Internet of People and Organizations" in contrast to the emerging "Internet of Things".

An interesting point to note is that the military market has long ago adopted hardware-based security solutions for many of the above-mentioned reasons; and as the Industry 4.0 and IoT deployment will also spread to critical infrastructure (electricity distribution, energy, petrochemical industry), it can be argued that the current military market security requirements of today will become the Industry 4.0 and IoT security requirements of tomorrow.

The rest of the white paper is structured as follows: Since security is fundamentally based on *cryptography* and its flawless implementation, Section 1 presents a brief overview of cryptography.

The benefits of hardware-based (specifically, FPGA-based) security are explained in Section 2, followed by an example of an FPGA-based communications endpoint implementation in Section 3. A critical component in a security design is the *Root-of-Trust*, and the FPGA-based root-of-trust solution is presented in Section 4, after which the white paper is concluded by the summary in Section 5.

Cryptography and Security

Cryptography is the art and science of protecting data and communication from unauthorized parties, typically referred to as adversaries. Cryptography is central for modern information security where cryptographic algorithms and protocols are used for numerous objectives: securing communication over untrusted networks, preventing unauthorized access to stored data, authenticating users, etc. In these systems, cryptography is used for ensuring specific security goals such as the ones shown in Table 1.

SECURITY GOAL	DESCRIPTION
Confidentiality (secrecy)	Ensures that information can be accessed only by authorized parties (e.g. the legitimate sender and receiver).
Integrity	Protects information from either accidental or intentional manipulation.
Authenticity	Provides assurances that an entity is the one who it claims to be or that data originates from its claimed origin.
Non-repudiation	Prevents an entity from denying its previous actions or commitments.

Table 1. Security Goals

Such security goals are ensured in computer systems using cryptographic protocols which in turn consists of cryptographic primitives, well-established algorithms for protecting specific security properties.

One way to categorize cryptographic primitives is by the type of keys they use:

- **Symmetric cryptography**: Cryptographic algorithms where the sender and receiver both have the same key.

  In symmetric encryption, the sender uses the key in a cryptographic algorithm to turn a plaintext (the message) into a ciphertext that an adversary cannot interpret without the key and, thus, protects the confidentiality of communication. The receiver uses the same key for decrypting the ciphertext back to the original plaintext. It is critical for the security that an adversary is not able to find the correct key. An example of a symmetric encryption algorithm is Advanced Encryption Standard (AES).

  In message authentication codes (MAC), the sender uses the key in a cryptographic algorithm to derive an authentication tag, which is sent to the receiver together with the message. The receiver uses the same key and the received message to compute another tag. The receiver accepts the message only if the two tags are the same (otherwise, the message or the tag are not the same that the sender sent). An adversary, who does not have the key, cannot forge a valid tag and, thus, the authenticity (and integrity) of the communication is protected. Hash-based MAC (HMAC) is an example of a MAC algorithm.

  Authenticated encryption combines confidentiality and authenticity protections into a single cryptographic primitive. An example is AES in Galois Counter Mode (AES-GCM).

- **Asymmetric cryptography**: Cryptographic algorithms where one party generates a pair of keys: private key and public key. The public key is generated from a private key using a mathematical function that is believed to be extremely hard[5] to invert and, thus, can be published (also to the adversary) without sacrificing security.

  In public-key encryption, the public key is used for encryption but decryption requires the private key; hence, everybody is able to encrypt, but only the receiver can open the encryption. Examples include RSA (named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman) and El-Gamal.

  In digital signatures, the private key is used for signing a message and the public key can be used for verifying the correctness of the signature; hence, everybody can use the public key to verify that a message was signed by the claimed signer and the signer cannot later deny this (non-repudiation). An example is Digital Signature Algorithm (DSA) or its elliptic curve variant ECDSA.

  A key exchange protocol uses asymmetric cryptography to share a secret key between two parties that can be later used in symmetric cryptography, which is typically much more efficient than asymmetric cryptography. Elliptic Curve Diffie-Hellman (ECDH) is an example of a key exchange protocol.

Note:

5. Extremely hard is to be understood as impossible in practice.

- **Others**: Certain cryptographic algorithms do not have a key at all.

  Well-known examples are cryptographic hash functions. A hash function takes in an arbitrarily long input and produces a fixed-length output, the hash, which can be seen as a fingerprint of the message. The hash function must be such that it is impossible to find the input message from the hash, another message that has the same hash as the input message, or two messages with same hash (a collision). Hash functions are used for numerous purposes in cryptographic protocols and are also used to build other cryptographic primitives such as MACs, key derivation functions (KDFs), etc. SHA-2 (Secure Hash Algorithm) is an example of a family of hash functions.

Modern cryptography is based on *Kerckhoffs's principle*: It is assumed that the adversary knows all the details of the cryptosystem except for the secret key. Hence, the security relies solely on the secrecy of the key. This means that for both symmetric and asymmetric cryptography it is essential that (secret) keys are hard to predict. Hence, the cryptographic system must have a source of randomness for deriving random keys. These random number generators (RNGs) are divided into two categories: true random number generators (TRNGs) and pseudo-random number generators (PRNGs). A TRNG uses a physical entropy source to derive truly random bits based on certain unpredictable physical phenomenon, whereas PRNGs are deterministic algorithms that generate a sequence of random-looking bits from an initial seed value. Typically, a PRNG is used even with a TNRG to clear possible biases from the bits from a TRNG.

A typical cryptographic protocol utilizes multiple cryptographic primitives. E.g., the Transport Layer Security (TLS) handshake protocol uses asymmetric cryptography for validating certificates with digital signatures and for key exchange to derive a shared secret key for a client and server. Then, the TLS record protocol uses symmetric cryptography for encryption and MAC (or authenticated encryption) for securing confidentiality and integrity of the bulk communication between the client and server.

Examples of cryptographic protocols are presented later in Sections 3: Examples and 4: Root-of-Trust.

FPGA Advantages vs. Processors

The majority of current security implementations are based on implementing the cryptographic protocols in software, most often using a third-party cryptographic software library, which is compiled for and executed on a general-purpose processor running an established and well-known operating system.

While software-based implementations of cryptography are most widespread, there is an increasing trend to implement security directly in hardware, especially in critical embedded systems. The motivations and realizations of this trend are explored further in the following sections.

The overall complexity of modern software-based security implementations presents multiple potential targets for a malicious third party (in technical terms, the "attack surface" is large), including but not limited to:

- The operating system;
- Device drivers;
- Implementation of cryptographic primitives (for example, with 3rd party cryptographic libraries);
- Compiler optimizations and possible microarchitectural changes between processor generations;
- The sheer depth of the software stack;
- Cache and memory management;
- Key management — for example, a buffer overflow bug can leak everything, including the secret keys;
- Lack of full control of the security algorithm implementation.

In addition to the above list, a software-based security implementation may also not meet the required performance (throughput and/or latency). In many cases the power consumption can also be a challenge.

Importantly, the continuous updateability (during the entire lifetime of the industrial system) for both software libraries (including the cryptographic libraries) as well as operating systems may be an insurmountable maintenance challenge. For example, the bug fixes[6] have to be updated to the IoT devices during its entire lifecycle, which can span several decades. This is a huge undertaking, and increases substantially the total cost of ownership of a software-based security solution.

A recent development in security thinking has been to also question the security of the underlying processor architecture. While previously the underlying hardware processor was automatically assumed to be secure, the recent disclosures about the possibilities to exploit performance-enhancing optimizations (for example, out-of-order execution and speculative branch execution) of processor microarchitecture (Meltdown, Spectre, Foreshadow, etc.) have put this basic assumption into question. Most of the security weaknesses mentioned before have been patched, but the possibility of new disclosures cannot be eliminated.

For the reasons described above a hardware-based security solution is often required, and a growing trend is to implement security in reprogrammable hardware, especially FPGAs.

Note:

6. For example, a search for "openssl" at cve.mitre.org (which maintains a list of publicly known cybersecurity vulnerabilities) gives 288 results (March 11, 2019).

FPGAs are integrated circuits, whose functionality is typically specified by using a Hardware Description Language (VHDL and/or Verilog), and which can be reprogrammed by the designer or the customer after deployment in the field. The vast majority of FPGAs available in the market today are based on volatile SRAM technology, meaning that the functionality of an FPGA is specified by a configuration file in an external non-volatile memory. Modern FPGA families support encrypted and authenticated configuration, where the configuration file is never stored as cleartext in the external non-volatile configuration memory, and at FPGA startup time the FPGA decrypts and authenticates the configuration file contents.

The traditional use case of reprogrammable logic (FPGAs) in security implementations has been to offload a host processor system by accelerating the performance-critical algorithms of a cryptographic protocol. The design techniques typically used to achieve speed-up include pipelining, parallelizability, and loop unrolling. The speedups — and also cost and power savings — have been especially true for symmetric cryptography, for example AES.

However, as mentioned earlier in this section, the advantages of FPGA-based security implementation are not limited to performance boosts, as properly implementing the security functions in FPGA logic also enhance the security level of the end product.

The additional technical advantages of FPGA-based implementation of cryptography include:

- Algorithm and protocol agility and updateability;
- Possibility to utilize built-in FPGA security features:
  - Encrypted and authenticated configuration;
  - Anti-tamper features;
  - Partial reconfiguration (in selected cases);
  - Design methods to achieve red-black separation in hardware;
- Tighter control of the algorithm implementation, including importantly key management.

In practice, an FPGA-based implementation of a security protocol utilizes a combination of individual intellectual property (IP) blocks. An example is presented in the following section.

Examples

As mentioned in Section 2, setting up a secure communication channel requires a complete cryptographic protocol built from multiple cryptographic primitives. Therefore, an FPGA-based implementation requires a combination of IP blocks for the cryptographic primitives, secure key storage for the keys used in the protocol, and control. In this section, an example of how an FPGA can set up a secure connection to a remote server over an insecure network (Internet) is presented, along with the IP blocks required. The focus is particularly on industrial applications where the throughput requirements are from low to moderate, but the required FPGA resource requirements should be as small as possible.

There are multiple ways to set up a secure connection, but this example assumes a TLS-like use case where the FPGA acts as a client in the protocol. The client must verify the authenticity of the server so that it can be assured that it connects to the real server and not to a malicious third party (a man-in-the-middle attack). It can be assumed that the client has the server's long-term public key that it can use for authenticating the server's messages with digital signatures[7]. The key exchange should derive different keys for each session (so-called *ephemeral keys*) that are not connected to any long-term keys to achieve *forward secrecy* that protects previous communications against any future compromises. After the key exchange has been completed, the bulk communication should protect confidentiality, integrity, and authenticity of the exchanged messages using the session keys.

The above can be realized with the following combination of FPGA-based IP blocks:

1. An RNG, preferably a TRNG to enable the secure seeding of a PRNG.
2. A hash algorithm with multiple uses in the protocol, including
   - PRNG of the RNG;
   - MAC algorithm (one example is HMAC) for protecting integrity of the bulk communication;
   - KDF which can be instantiated with a hash algorithm (one example is the HMAC-based HKDF).
3. Asymmetric cryptography for digital signatures and key exchange, where the recommended way is to use an IP block for Elliptic Curve Cryptography (ECC), since the other main option, RSA, is significantly less efficient. Operations for asymmetric cryptographic are the computationally heaviest operations in building a secure connection. However, as these operations are needed only in the beginning of a session, their latency is not critical and IP blocks, where the FPGA resource utilization has been minimized, can be selected.
4. Symmetric cryptography. This is used for protecting the bulk communication. The most popular choice here is to use AES in a secure mode of operation, for example AES-GCM which provides simultaneous protection for confidentiality, integrity, and authenticity (so-called authenticated encryption).
5. Secure key storage used for storing all security-critical information in the FPGA.

Note:

7. FTLS uses certificates issued by mutually trusted parties (certificate authorities) to build trust for the public keys, but without loss of generality the example assumes that a trusted public key exists in the FPGA

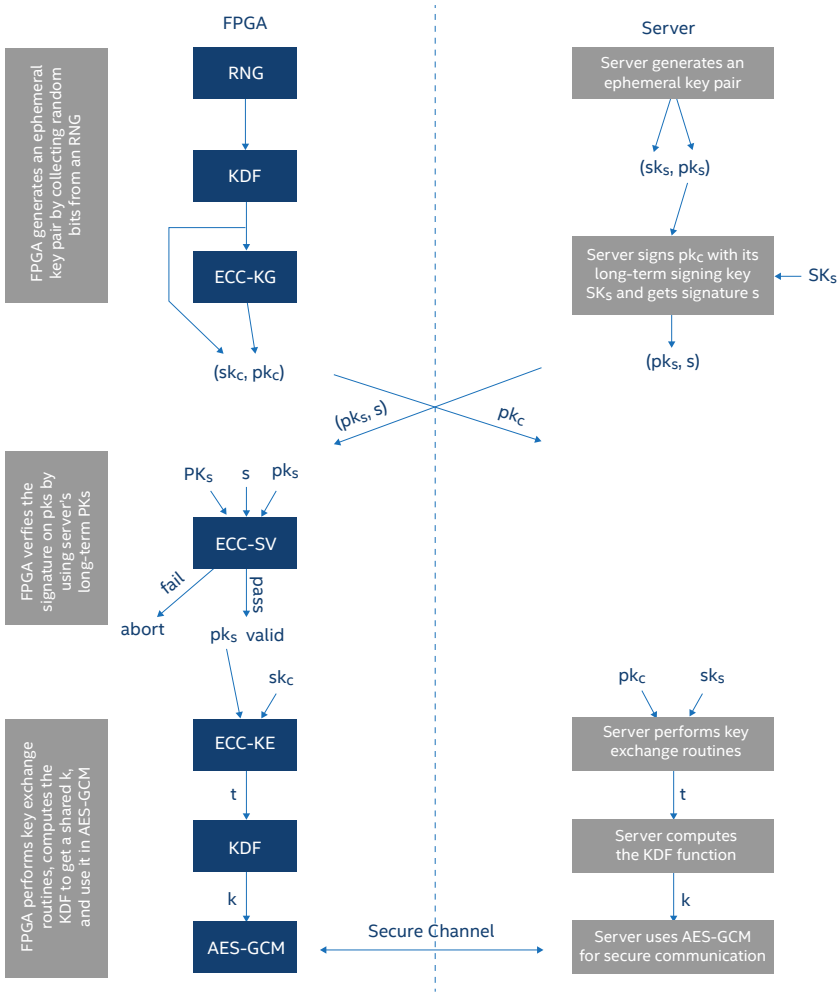

Figure 1. Setting up a secure connection between an FPGA client and a remote server

The combination of IP blocks is used for setting up a secure connection to the remote server in the following way (see Figure 1 for a pictorial presentation):

1. The FPGA uses the RNG, KDF (the hash algorithm IP block), and the ECC IP block to generate an ephemeral key pair (the client's secret key *skc* and public key *pkc*). The public key *pkc* is sent to the server.
2. The server also generates its own ephemeral key pair (the server's secret key *sks* and public key *pks*). It signs this ephemeral public key *pks* by using its long-term secret key SK and sends *pks* and signature s to the client.
3. The FPGA uses the asymmetric cryptography IP block based on ECC for verifying the signature s attached to the server's ephemeral public key *pks*. This verification is done by using the server's long-term public key PK that the FPGA has and if this verification succeeds, the FPGA can be assured that the ephemeral public key indeed came from the legitimate server and not from an adversary.
4. The FPGA uses its own ephemeral secret key *skc* and the server's ephemeral public key *pks* to calculate a shared secret value t with the asymmetric cryptography IP block for ECC-based key exchange. This shared secret value is then fed into the KDF (the hash algorithm IP block) to derive a shared secret key k that can be used in the bulk communication.
5. The server uses its own ephemeral secret key *sks* and the client's ephemeral public key *pkc* for calculating the same shared secret key k (with key exchange computations and KDF).
6. The FPGA uses shared secret key *k* and AES-GCM (the symmetric cryptography IP block) for secure communication with the server. The RNG may be used for generating initialization vectors used in AES-GCM. Once the session is closed, all ephemeral keys (including the shared secret key) are removed.

In the above steps, only the FPGA (client) verified the authenticity of the server. The FPGA could be authenticated similarly with long-term public keys in steps 1 and 2 or with another authentication mechanism over the secure communication channel that was set up above. This approach requires root-of-trust in the FPGA, which is the subject of the following section.

Root-of-Trust

A root-of-trust can be defined as a set of functions in a trusted computing unit, which can always be trusted by the system. Consequently, a root-of-trust enables building trust to the entire system. Additionally, it is critical that every device in the system has a unique identifier or a secret key, which can be used to derive additional key material, including public keys.

The need for hardware-based root-of-trust has been recognized for critical IoT applications, and the most common way to achieve this in embedded systems has been to use an external security chip, also often called a Trusted Platform Module (TPM). However, external security chips may not always have the features required by the application (for example, their symmetric encryption throughput may not be sufficiently high), In addition, they may also add cost, board space, and power consumption to the design budget.

For designs already using a volatile FPGA, it would be tempting to embed a unique secret key into the encrypted configuration file to attempt hardware-based root-of-trust functionality. However, this approach requires the manufacturer to first generate unique configuration files for each FPGA-based end product, and afterwards to maintain a post-launch database linking every individual end product to a particular configuration file, even though the functionality of all launched products is based on the same FPGA design. Very quickly the cost and time required for these configuration file generation and maintenance tasks would become unbearable.

To overcome this challenge, Xiphera has developed an FPGA-based solution for root-of-trust functionality requiring only an FPGA and its configuration memory[8]. The Xiphera root-of-trust solution is based on currently available Xiphera IP blocks and allows using the same configuration file for a batch[9] of FPGAs.

The main target application for an FPGA-based root-of-trust solution is in new designs and deployments of FPGA-based security. However, the root-of-trust solution can also be retrofitted to existing installations in selected cases, provided that there are enough available resources in an FPGA and its reprogramming can be performed.

The functionality of FPGA-based root-of-trust is accomplished by using a combination of IP blocks for RNG, KDF, and secure key storage, guaranteeing that all security-critical functionality remains inside the FPGA. The root-of-trust solution provides strong separation of security-critical key material and the general data, ensuring that keys cannot be accessed from the outside, and that the keys are never outside the FPGA without being protected by strong encryption.

It should be noted, that the FPGA-based root-of-trust solution allows for integration with a cipher suite of the customer's own choosing or as required by a particular standard. Likewise, the performance (for example, throughput) of a cryptographic algorithm can be customized. As an example, if the bulk communication protocol uses AES-GCM as the authenticated encryption algorithm, the throughput can be set to 100 Mbps, 1 Gbps, or 10 Gbps, but all versions are based on the same root-of-trust solution.

The FPGA-based root-of-trust solution can be interfaced with external control logic and/or a host processor, which can be either external or internal to the FPGA (softcore or hardcore). This makes sense for handling the non-security critical parts of a communications protocol, for example TLS over TCP/IP.

Summary

Industry 4.0 is the ongoing and accelerating fourth Industrial Revolution, but its ultimate adoption and success require comprehensive end-to-end security. The vast majority of security breaches so far have been due to the careful exploitation of weaknesses in software-based implementations of security and cryptography. Therefore, implementing the critical functionality directly in hardware is being recognized as the preferred methodology to meet the critical security requirements. Additional advantages of hardware-based security implementation may also include improvements in performance and reduction in power consumption.

FPGAs are a strong candidate for hardware-based security designs, as their reprogrammability enables various iterations of the cryptographic algorithms and security protocols during the design phase, while allowing the designer to maintain full control and understanding of the implementation. Typically, FPGA-based security uses a combination of individual IP blocks to build complete security protocols, including key management.

Xiphera's root-of-trust solution requires only a volatile SRAM-based FPGA and external configuration memory. The root-of-trust solution allows the FPGA to generate a unique secret key and to derive additional secret and public keys from it during initialization so that the same keys are restored even if the power is turned off and on again. This makes it possible to build a secure hardware root-of-trust module with a volatile FPGA.

Notes:

8. For the Intel® MAX® 10 FPGA, the Xiphera root-of-trust solution can be a single-chip solution as the Intel MAX 10 FPGA configuration flash is in the same package.
9. Batch can be thought of as a group of FPGAs having the same configuration; this can be customer-specific, product-specific, etc., but the size of a batch does not have practical numerical upper or lower boundaries.

Glossary

Attack surface. The different points of a system that an adversary can target in order to attack the system.

Authenticated encryption. An encryption system that provides both confidentiality (secrecy) and authenticity (e.g., AES-GCM).

Buffer overflow. An event where data are written (or read) over the boundaries of a buffer.

Configuration file. The file that configures an FPGA to implement a specific design.

Cryptanalysis. The study of cryptosystems that aims to break the protection offered by cryptography.

Cryptographic algorithm. A specific algorithm for performing a cryptographic operation (e.g., AES).

Cryptographic protocol. A protocol that applies specific cryptographic algorithms to achieve certain security goals (e.g., TLS).

Cryptography. The science and techniques of protecting data and communication from unauthorized parties.

Digital signature. A technique that allows verifying the authenticity of a digital document, cf. handwritten signatures for paper documents.

Elliptic curve cryptography. Asymmetric cryptosystems based on the use of mathematical constructions called elliptic curves. Their security relies on the difficulty of elliptic curve discrete logarithm.

Encryption/decryption. The processes of encoding and decoding the context of a message so that it can be read only by the sender and receiver.

Ephemeral key. A cryptographic key that is generated and used only for a single session.

Field programmable gate array. A programmable integrated circuit that enables configuring the operation of the device at the logic level.

Forward secrecy. A feature that assures that previously used session keys are not compromised even if long-term secrets are compromised.

Hash function. A function that maps an arbitrarily long message to a fixed-length value (the hash).

Intellectual property block. A component of a digital design that implements a specific functionality (e.g., an implementation of a cryptographic algorithm).

Key exchange. A technique that allows two parties to exchange a secret key over an insecure network.

Key management. The process of managing (generating, storing, using, etc.) cryptographic keys in a cryptosystem.

Message authentication code. A technique that authenticates a message and prevents it from being tampered.

Random number generator. A component or function that produces random bits. A true random number generator uses a physical entropy source based on certain unpredictable physical phenomenon and pseudo random number generator is a deterministic algorithm producing random-looking bit sequence from a seed value.

Red-black separation. Strict separation of unencrypted (red) and encrypted (black) information.

Root-of-trust. A set of (cryptographic) functions that can always be trusted by the system.

Transport Layer Security (TLS). A protocol that provides a secure communication channel between a client and a server over Internet.

WP-01297-1.0