Energy and Execution Time Analysis of a Software-based Trusted Platform Module *

Najwa Aaraj†, Anand Raghunathan‡, Srivaths Ravi⋆, and Niraj K. Jha†
† Department of Electrical Engineering, Princeton University, Princeton, NJ 08544
‡ NEC Laboratories America, Princeton, NJ 08540
⋆ Texas Instruments R&D Center, Bangalore, India
†{naaraj, jha}@princeton.edu ‡anand@nec-labs.com ⋆srivaths.ravi@ti.com

Abstract

Trusted platforms have been proposed as a promising approach to enhance the security of general-purpose computing systems. However, for many resource-constrained embedded systems, the size and cost overheads of a separate Trusted Platform Module (TPM) chip are not acceptable. One alternative is to use a software-based TPM (SW-TPM), which implements TPM functions using software that executes in a protected execution domain on the embedded processor itself. However, since many embedded systems have limited processing capabilities and are battery-powered, it is also important to ensure that the computational and energy requirements for SW-TPMs are acceptable.

In this work, we perform an evaluation of the energy and execution time overheads for a SW-TPM implementation on a Sharp Zaurus PDA. We characterize the execution time and energy required by each TPM command through actual measurements on the target platform. In addition, we also evaluate the overheads of using SW-TPM in the context of various end applications, including trusted boot of the Linux operating system (OS), secure file storage, secure VoIP client, and secure web browser. Furthermore, we observe that for most TPM commands, the overheads are primarily due to the use of 2048-bit RSA operations that are performed within SW-TPM. In order to alleviate SW-TPM overheads, we evaluate the use of Elliptic Curve Cryptography (ECC) as a replacement for the RSA algorithm specified in the Trusted Computing Group (TCG) standards. Our experiments indicate that this optimization can significantly reduce SW-TPM overheads (an average of 6.51X execution time reduction and 6.75X energy consumption reduction for individual TPM commands, and an average of 10.25X execution time reduction and 10.75X energy consumption reduction for applications). Our work demonstrates that ECC-based SW-TPMs are a viable approach to realizing the benefits of trusted computing in resource-constrained embedded systems.

1 Introduction

Security is emerging as an important consideration in the design of many embedded systems. Embedded systems are often used to perform safety-critical functions or financial transactions, and capture or store our personal data. On the other hand, they feature increasing software complexity, and are often deployed in physically insecure spaces or connected to public networks, exposing them to the same, if not greater, risk factors and security threats faced by PCs and the Internet. Our experiences with general-purpose computing systems indicate that conventional reactive approaches to security, such as anti-virus and anti-spyware tools, OS and application patches, *etc.*, are not effective in preventing security attacks (the number of reported attacks as well as newly discovered vulnerabilities continue to grow each year [1]). Therefore, it is important to build-in security during various stages of the embedded system design process [2].

Trusted computing is an emerging paradigm that addresses information security concerns in a wide variety of computing systems. Trusted computing standards are driven by the computing and communications industries through the TCG [3]. The conventional approach to trusted computing requires several hardware and software enhancements to the target system, including the addition of a separate chip called the TPM that is affixed within the target system.

The TPM acts as a root of trust for the system that contains it, providing capabilities for secure storage, secure reporting of platform configuration measurements, and cryptographic key generation, among other functions (an overview of TPM functions is provided in Section 2.1). In addition to the above functions, the TPM chip implements tamper-resistance techniques to prevent a wide range of physical and hardware-based attacks. TPM chips are being produced by several vendors, and have been incorporated into several desktop and laptop PCs, and servers [4, 5]. It is predicted that by 2010, worldwide shipments of TPM modules in PC client systems will reach more than 250 million [6]. In addition to the use of the TPM envisioned by the TCG, researchers have demonstrated its use in enhancing the security of a variety of software applications and services [7, 8, 9, 10, 11, 12].

Given the benefits and growing use of trusted computing in general-purpose systems, it is natural to explore the application of these concepts and technologies to enhance the security of embedded systems. Recent efforts to adapt trusted computing standards to resource-constrained environments include the TCG's Mobile Phone Working Group [13] and the Trusted Mobile Platform Alliance [14]. However, the hardware enhancements, including the addition of the TPM chip, may impose an unacceptable overhead in the context of cost- and size-constrained embedded systems. For such systems, we propose the use of a software implementation of the TPM functions (referred to as software TPM or SW-TPM) in order to enable the adoption of trusted computing techniques. Although not completely equivalent to a conventional TPM chip in terms of protection against physical and hardware attacks, SW-TPM can be executed within protected or isolated execution domains that are increasingly provided by embedded CPUs (*e.g.*, ARM TrustZone [15]), and can utilize on-chip storage in order to provide a reasonable degree of tamper-resistance. However, one issue that remains is whether the computational and energy requirements to perform the TPM functions are acceptable. In this work, we address this question by analyzing and optimizing the energy and execution time overheads imposed by SW-TPM. Our work makes the following contributions:

- We perform a comprehensive characterization of SW-TPM running on a battery-powered handheld device (Sharp Zaurus PDA), and measure the execution time and energy requirements for various TPM commands.
- We evaluate the overheads imposed by using TPM functions in end applications, including trusted boot of the Linux OS, secure file storage utility, secure VoIP client, and secure web browser.
- In order to alleviate the overheads imposed by SW-TPM, we propose and evaluate the use of ECC as a replacement for the RSA algorithm specified in the TCG standards. Our experiments indicate that this results in a substantial reduction in SW-TPM overheads.

In a broader sense, our work demonstrates the feasibility of using SW-TPM to realize the benefits of trusted computing in resource-constrained

*Acknowledgments: This work was supported by NSF under Grant No. CCR-0326372.

978-3-9810801-2-4/DATE07 © 2007 EDAA

embedded systems.

The rest of the paper is organized as follows. Section 2 presents background material on TPM functions. Section 3 introduces the concept of SW-TPM, and details the different characterizations of SW-TPM commands. Section 4 discusses different applications, their trusted implementations, and the incurred energy and execution time overheads. For the ease of presentation and comparison, experimental results are included in Sections 3 and 4 for both the original SW-TPM and the optimized SW-TPM that uses ECC-based asymmetric cryptography.

2 Background

In this section, we provide an overview of the functions performed by the TPM and summarize its various commands.

2.1 TPM Overview

Three roots of trust lie at the core of a trusted platform: (i) a Root of Trust for Measurement (RTM), which is responsible for taking platform integrity measurements, (ii) a Root of Trust for Storage (RTS), which securely stores different integrity measurements, and (iii) a Root of Trust for Reporting (RTR), which is responsible for reliably reporting values stored in the RTS. The TPM chip includes the RTR and RTS functions, and depending on the implementation, it may also include parts of the RTM. The TPM also supports other functions such as cryptographic key generation, and data sealing and binding.

Figure 1 describes the various layers of a trusted platform, and outlines the different components of the TPM.

The protected storage in a TPM includes the Platform Configuration Registers (PCRs), and limited volatile and non-volatile storage spaces. Protected storage is accessible only within the TPM, and is shielded from physical attacks. Each PCR is 20 Bytes in length, and can store an SHA-1 digest that represents the platform integrity measurements. PCR values are updated by concatenating their original values with the new measurements, followed by an SHA-1 operation (performed by the TPM's SHA-1 engine) whose output is written back to the PCR.

TPM's Random Number Generator, Key Generator and RSA Engine components support the generation of RSA keys as well as symmetric keys, and RSA cryptographic operations, such as signing and encryption/decryption. TPM keys can be classified into different categories. Each TPM is shipped with a 2048-bit RSA key stored inside it. This key, known as the Endorsement Key (EK), is generated by the vendor or manufacturer, and stored in the non-volatile memory within the TPM. EK should never be disclosed and is only used for establishing the TPM ownership, *i.e.*, creating the Storage Root Key (SRK), and for issuing Attestation Identity Keys (AIKs). The SRK is the root of a hierarchy of TPM keys, and is used for encrypting other migratable keys that are stored outside the TPM. AIKs are special-purpose 2048-bit RSA keys; their private portion is securely stored by the TPM and their public part, along with an AIK certificate, is used for platform attestation and authentication, and for key certification. The TPM is also capable of generating an arbitrary number of user keys that can be classified as either migratable or non-migratable.

Protected storage and cryptographic functions are crucial for platform attestation and integrity checking. Whenever a platform is challenged to prove its integrity or the integrity of any application running on it, the RTR (together with the software that uses it) is responsible for reading relevant PCR values, signing them with the private portion of an AIK, and sending the signatures, along with the public portion of the AIK and necessary credentials, to the challenger. The challenger verifies the authenticity of the platform by checking the credentials and comparing the measured values with expected ones.

The path between applications and the TPM is handled by a set of software components called the TPM Software Stack (TSS). The main objective of the TSS is to manage TPM resources as well as provide access to the TPM. The different components of the TSS are listed in Figure 1. We refer interested readers to [13] for further details.

2.2 TPM Commands

A number of commands control the activities performed within the TPM, and enable software running on the platform to benefit from the set of security features that it provides. We divide these commands into eight categories, each of which is summarized below (a full listing of TPM commands can be found in [13]).

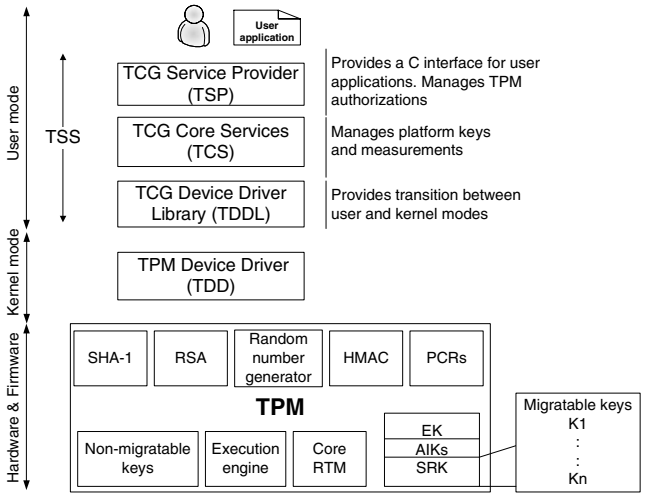

Figure 1: Modules and interfaces of a trusted platform

- Authentication commands: used for creating session authentications that allow interactions with the TPM (TPM_OIAP and TPM_OSAP).
- Capability commands: used for getting TPM information (TPM_GetCapability).
- Cryptographic commands: used for digital signatures (TPM_Sign) and random number generation (TPM_GetRandom).
- Identity commands: used for creating AIKs (TPM_MakeIdentity), and decrypting AIK credentials (TPM_ActivateIdentity) obtained from a trusted third party (TTP).
- Measurement commands: used for reading PCR contents (TPM_PcrRead), signing (TPM_Quote) and extending PCR values (TPM_PcrExtend).
- Ownership commands: used for establishing ownership of the TPM, and creating the SRK (TPM_TakeOwnership).
- Start-up commands: used for starting, enabling, resetting, and saving the state of a TPM (tpm_init_data and TPM_Startup).
- Storage and key management commands: used for key generation (TPM_CreateWrapKey), key loading into TPM memory space (TPM_LoadKey), key eviction (TPM_EvictKey), data binding/unbinding (TPM_UnBind), *i.e.,* encrypting/decrypting data with a key managed by the TPM, data sealing (TPM_Seal), *i.e.*, encrypting data and associating it with specific PCR values, and data unsealing (TPM_Unseal), *i.e.*, decrypting data if the PCR values match those at the time of sealing. These commands operate on TPM keys of sizes up to 2048 bits.

3 Characterization of the Software TPM

The TPM security features outlined in Section 2.1 are very useful in many embedded systems. However, due to area and cost constraints, some embedded systems cannot be augmented with a conventional TPM chip. Therefore, we explore the feasibility of a SW-TPM, which performs the same functions as a hardware TPM, *i.e.*, supports all the three roots of trust, as well as other cryptographic capabilities.

Although SW-TPM does not provide the same security level as a TPM chip, particularly with respect to physical attacks, executing the SW-TPM in a protected execution domain of the CPU (*e.g.*, ARM TrustZone), and using on-chip memory, provides resistance to software attacks, including compromises of the OS, and a limited number of physical attacks. In fact, this is suggested in [14] as an acceptable implementation of the TPM.

This section describes our implementation of the SW-TPM and all supporting software running altogether on an embedded platform. It also describes the experimental set-up used to perform energy and execution time characterization of SW-TPM, and the results obtained therefrom.

3.1 SW-TPM Implementation

Our SW-TPM implementation is adapted from the public domain TPM emulator [16]. This emulator provides basic TPM functions, such as

RSA cryptography and HMAC and SHA-1 hashing functions, and provides several TPM commands, some of which were listed in Section 2.2. Our changes to the emulator include the following:

- Random number generation: We use a hash-complemented Mersenne Twister (MT) random number generator [17], *i.e.,* we run the output of MT through SHA-1.
- ECC: SW-TPM supports ECC in the binary field GF(2^m). The deployment of ECC on our embedded platform is due to its small key sizes for offering the same security robustness as RSA. Hence, it requires less resources such as processor cycles and energy.

  ECC-enabled SW-TPM supports key generation and validation, digital signature generation and verification, encryption, and decryption. ECC key sizes supported are 224 bits (equivalent to 2048-bit RSA keys), 192 bits (although 192 bits do not represent an equivalent to any RSA key size, we choose to study its performance when used with SW-TPM because it is widely used in security applications), and 160 bits (equivalent to 1024-bit RSA keys).
- AES_CBC cryptography: SW-TPM supports the Advanced Encryption Standard (AES) algorithm, running in Cipher Block chaining (CBC) mode. This engine is specifically used for ECC encryption and decryption, and for decrypting AIK credentials.

For brevity, we do not delve further into the details of the TPM emulator Linux implementation, and we refer interested readers to the documentation available in [16].

3.2 SW-TPM Characterization

This section first describes the experimental set-up used for evaluating SW-TPM's execution time and energy requirements. It then reports results obtained for various TPM commands performed using SW-TPM.

3.2.1 Experimental Set-up

The experimental set-up used in our experiments is shown in Figure 2.

A) Hardware: The embedded system used is a Sharp Zaurus SL-5600 [18] running Embedix and Qtopia. It is equipped with a 400MHz Intel Xscale processor (PXA-250), with a 32MB SDRAM and 64MB Flash ROM. Energy measurements were performed using an Agilent 34401A digital multimeter interfaced with a 3.2GHz Intel Pentium IV PC running Windows. Throughout the execution of the program of interest on the PDA, we probed the voltage drop across a 0.1Ω sense resistor that is connected in series with the 5V DC power supply cord to the PDA. A data acquisition program, written in Visual C++ on Windows, controlled the multimeter to sample the voltage drop across the resistor at 25Hz. A MATLAB script calculates the energy consumption by integrating the power time-series using the trapezoidal rule.

B) Software: SW-TPM commands are adapted from the open-source software-based TPM emulator [16], which is implemented as a Linux kernel module. The code was ported from the x86 architecture and Redhat Linux to the PXA-250 processor and Embedix (changes were required due to the differences in the kernel, as well as architectural parameters such as endianness). We also ported the GNU MP library onto the system, in order to use the RSA, HMAC, and SHA-1 functions provided by the emulator. In order to implement the ECC-enabled SW-TPM, we (i) developed an ECC engine that uses the Big Number (BN) library provided by OpenSSL [19], and the Montgomery [20] based point multiplication module integrated into OpenSSL by Sun microsystems [21], and (ii) changed the command processing code in SW-TPM in order to interact with the ECC engine instead of the RSA engine. We also used two other open-source software packages: TrouSerS and Testsuite [22]. TrouSerS is an open-source implementation of the TSS, and Testsuite provides different test cases that exercise a TPM through the TSS. We installed the two packages, along with the TPM emulator, on an x86 Linux PC. We ran different test cases, and probed the input and output parameters being fed to and generated from the TPM emulator (input parameters are arguments necessary for the execution of a command, and output parameters are the results of command execution). The input parameters were then applied to the SW-TPM implementation running on the PDA, and the execution time and energy measurements were performed. For commands not supported by Testsuite, such as TPM_MakeIdentity and TPM_ActivateIdentity, we had to provide our own parameters. In order to exercise the ECC Engine, RSA-dependent input parameters were changed to the corresponding ECC counterparts.

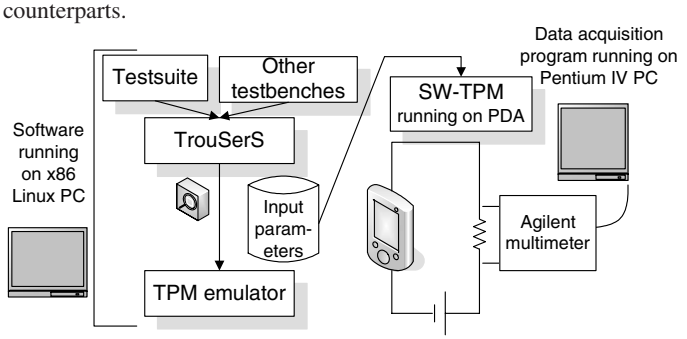

Figure 2: Hardware and software experimental set-up

3.2.2 Measurement Results

In this section, we present the execution time and energy consumed by SW-TPM on the PDA in order to execute various TPM commands. Results are presented for the original RSA-based SW-TPM, as well as the proposed ECC-based SW-TPM. For some commands, namely, the commands categorized as the storage and key management commands, and TPM_Sign, several measurements were performed for different key sizes (we give results for the 2048-224 and 1024-160 equivalent RSA-ECC key sizes, as well as for 512-bit RSA keys, and 192-bit ECC keys). Also, for commands that process user data, the data size is varied. The results of our experiments are reported in Table 1. Column 1 represents the command executed. Columns 2-3 give the key size (K) and data size (D). K (D) is indicated as n/a for commands that do not involve cryptographic operations (do not process user data). Column 4 gives energy measurements in milliJoules (mJ), and column 5 reports the execution times for the TPM commands in milliseconds (msec.). The results indicate that commands involving RSA operations, particularly private key operations, which require manipulation of large numbers, and a resource-consuming modular exponentiation, impose a high execution time overhead. For instance, the TPM_MakeIdentity command, which involves 2048-bit RSA key generation and validation, as well as encryption of the private AIK using the SRK, in addition to other cryptographic functions, takes 29.63 sec. and consumes 70.94 J of energy. Similarly, large execution times and energy consumptions are required for TPM_TakeOwnership, TPM_CreateWrapKey, TPM_Unseal, *etc*. However, this overhead is reduced by using ECC: execution time and energy requirements for the TPM_MakeIdentity command are reduced to 2.43 sec. and 5.86 J, respectively. In fact, by using ECC, we could achieve an average reduction of 6.51X and 6.75X for execution time and energy, respectively, across all commands.

Table 2 presents macromodels that capture the energy for the TPM_Sign and TPM_Seal commands as a function of the key size K and data size D. K can vary up to 2048 (224) bits for RSA (ECC), and D assumes values up to 144 Bytes. As can be concluded from the macromodels, and the numbers reported in Table 1, energy and execution time requirements vary more significantly with the key size rather than with the data size (especially when RSA cryptography is used).

Note that, in order to account for uncontrollable variables, such as the randomness of the keys, and to minimize measurement error for commands that require small running times, the results presented are based on the average of several executions of each command (an average of 16 executions for a single command).

While the data presented in this section are useful for evaluating the requirements of SW-TPM in isolation, it is also important to place these overheads in the context of actual applications. In the next section, we propose trusted extensions for several applications and study the impact of using SW-TPM on their execution time and energy consumption.

4 SW-TPM in User Applications

This section considers the usage of SW-TPM in the context of four different applications. We describe the trusted extensions of these applications, and evaluate the effect of these extensions in terms of energy and execution time. The results of our experiments are presented in Table 3.

Column 1 indicates whether RSA-enabled SW-TPM or ECC-enabled SW-TPM is used. Columns 2-3 report energy and execution time for the untrusted application, and the total overhead required by the trusted version of this application, respectively. Columns 4-5 give the energy and execution time overheads due to the executed SW-TPM commands. The results, discussed in the context of each application in the rest of this section, indicate that the executed SW-TPM commands, within the applications, require an average of 10.75X less energy and an average of 10.25X less execution time when ECC, instead of RSA, is enabled.

4.1 Trusted Boot

Trusted boot of the OS is one of the most common applications of trusted platforms [3]. We extend the standard booting sequence of the Embedix OS on the Sharp Zaurus PDA into a trusted version in order to monitor its integrity. We implement a simplified version of the trusted boot sequence suggested for enhancing the Linux bootloader GRUB [23]. This application is not fully implemented yet, however, the fine-grained analysis of the SW-TPM commands in Section 3.2 helps us estimate the execution time and energy overhead of the final application.

- The PDA's bootloader is divided into two stages for the purpose of trusted boot. At power on, stage 1 of the bootloader executes. This stage is assumed to be trusted, *i.e.*, it acts as the Core Root of Trust for Measurement (CRTM). It sets SW-TPM to the active state, and performs a TPM_Startup command. Stage 1 hashes stage 2 after loading it, and PCR 4 is extended.
- Stage 2 of the bootloader loads the kernel and performs an integrity measurement of its image, *i.e.,* it hashes the kernel image. PCR 5 is extended using the resulting hash value.
- The kernel hashes the pre-installed kernel modules, kernel configuration files, and pre-installed application programs. PCR 8 is extended using the resulting hash value.

Once the PCRs are extended, boot integrity can be achieved by comparing PCR values with previous integrity measurements (*i.e.,* hash computations) taken under an untampered normal boot sequence.

In summary, the SW-TPM commands executed during the trusted boot process are: TPM_TakeOwnership, tpm_init_data, TPM_Startup, and TPM_PcrExtend. In addition, further overhead is incurred by integrity measurements performed at each stage of the booting process. The energy and execution time measurements for trusted boot are reported in Table 3, rows 1 and 2.

4.2 Secure Storage

The secure storage capability provided by the TPM, in particular the sealed storage capability, is especially attractive for securing sensitive data, such as passwords, pins, biometric information, *etc.*, and preventing their usage by corrupted programs. In this section, we identify the execution time and energy needed for sealing a 500 Bytes file F containing biometric templates, which are used by an authentication program. The sealing operation proceeds as follows:

- Create a 2048-bit RSA key (224-bit ECC key) and load it into the SW-TPM memory space.
- Seal the biometric templates data to PCR 15: prior to sealing the data, the OS hashes the executable binary of the biometric authentication program, and its digest is placed in PCR 15. Then, the biometric data is sealed (144 Bytes at a time). Before unsealing the data, PCR 15 is reset and extended with a new hash value of the authentication program binary. The data can be retrieved only if PCR 15 values at the time of sealing and the time of unsealing match, *i.e.*, if the authentication program has not been corrupted from the time the data was sealed.

The SW-TPM commands executed are: TPM_OSAP, TPM_CreateWrapKey, TPM_OIAP, TPM_LoadKey, TPM_Terminate -Handle (this command has not been included in Table 1 because its execution time on the PDA is insignificant), TPM_PcrExtend, TPM_PcrRead, and TPM_Seal. In addition, further overhead is incurred by integrity measurements of the authentication program. The energy and execution time measurements for the secure storage application are given in Table 3, rows 3 and 4.

4.3 Secure Voice over Internet Protocol (VoIP)

VoIP is a technology that uses a broadband Internet connection to place phone calls. Voice information is compressed using a Compressor/Decompressor (Codec) and sent in network packets over the Internet. VoIP can become the target of many security attacks, such as spam, call hijacking, denial of service attacks, *etc*. Hence, VoIP security measures are critical. Commercial VoIP security solutions have been deployed, and an end-to-end solution based on the TPM can be found in [8].

Table 1: Energy and execution time for TPM commands

Command	K (bits) ECC/RSA	D (Bytes)	PDA measurements Energy(mJ)	Time(msec.)
Authentication commands				
TPM_OIAP	n/a	n/a	0.61	0.21
TPM_OSAP	n/a	n/a	2.38	0.82
Capability commands				
TPM_GetCapability				
(Key info.)	n/a	n/a	0.10	0.04
(Manufacturer info.)	n/a	n/a	0.10	0.04
(PCR info.)	n/a	n/a	0.20	0.07
Cryptographic commands				
TPM_GetRandom	n/a	20	0.55	0.19
TPM_Sign	224/2048	20	450/2210	191/902
	224/2048	50	492/2221	204/926
	224/2048	100	531/2394	216/970
	160/1024	20	210/806	90/343
	160/1024	50	242/930	114/388
	160/1024	100	319/1006	131/409
	192/512	20	321/626	136/265
	192/512	50	350/656	148/274
	192/512	100	361/760	153/305
Identity commands				
TPM_ActivateIdentity	224/2048	n/a	598/12824	348/5239
TPM_MakeIdentity	224/2048	n/a	5859/70943	2425/29634
Measurements commands				
TPM_PcrRead	n/a	n/a	17.32	6.69
TPM_PcrExtend	n/a	n/a	32.28	12.46
TPM_Quote	224/2048	n/a	762/2475	381/1239
Ownership commands				
TPM_ReadPubek	224/2048	n/a	0.31/3.10	0.12/1.22
TPM_TakeOwnership	224/2048	n/a	5619/66777	2391/28619
Start-up commands				
tpm_init_data	224/2048	n/a	1.71/25.39	0.69/10.52
TPM_Startup	224/2048	n/a	0.48/1.46	0.19/0.58
Storage and key management commands				
TPM_CreateWrapKey	224/2048	n/a	5558/42582	2322/16938
	160/1024	n/a	4128/12133	1813/4594
	192/512	n/a	4419/8395	1880/3025
TPM_EvictKey	224/2048	n/a	8.36/37.08	3.31/14.78
	160/1024	n/a	6.70/16.62	2.71/6.62
	192/512	n/a	6.72/7.73	2.79/3.10
TPM_GetPubKey	224/2048	n/a	640/10592	229/4388
	160/1024	n/a	471/1504	157/567
	192/512	n/a	516/852	172/453
TPM_LoadKey	224/2048	n/a	810/14547	336/5367
	160/1024	n/a	593/4557	261/1796
	192/512	n/a	737/2092	301/841
TPM_Seal	224/2048	20	1103/3751	463/1476
	224/2048	50	1125/3785	472/1564
	224/2048	100	1313/4271	530/1761
	160/1024	20	769/1898	322/796
	160/1024	50	806/2195	334/967
	160/1024	100	819/2965	342/1178
	192/512	20	1001/1019	420/427
	192/512	50	1026/1063	431/481
	192/512	100	1093/1326	444/551
TPM_Unseal	224/2048	256	1444/14056	585/5520
	160/1024	256	952/4679	391/1880
	192/512	256	1279/1778	525/714
TPM_UnBind	224/2048	256	1459/10480	576/4103
	160/1024	256	974/4269	384/1699
	192/512	256	1284/1616	524/669

In this section, we describe a simple security scheme for VoIP that utilizes the TPM to authenticate the client to the server, ensure that the client application has not been corrupted, and exchange the symmetric key used for bulk (voice) data encryption. In order to illustrate this scheme, we consider the set-up shown in Figure 3. We installed the open-source Private Branch eXchange (PBX) server Asterisk [24] on a Linux machine, and two VoIP clients, based on the Inter-Asterisk eXchange (IAX) protocol, on a PDA and a laptop.

We assume that the PBX server is secured from tampering and plays the role of an authentication agent (AA), *i.e.*, it maintains a log of differ-

Table 2: Energy macromodels for the TPM_Sign and TPM_Seal

Command	Crypto type	Energy model (C + A*D + B*D^2 + X*K + Y*K^2(mJ))
TPM_Sign	ECC	31.269 + 1.434*D - 0.004*D^2 - 0.642*K + 0.011*K^2
	RSA	29.994 + 10.389*D - 0.061*D^2 + 0.349*K + 0.00029*K^2
TPM_Seal	ECC	6.415 - 0.067*D + 0.012*D^2 + 3.980*K + 0.005*K^2
	RSA	5.766 - 10.033*D + 0.142*D^2 + 2.435*K - 0.00026*K^2

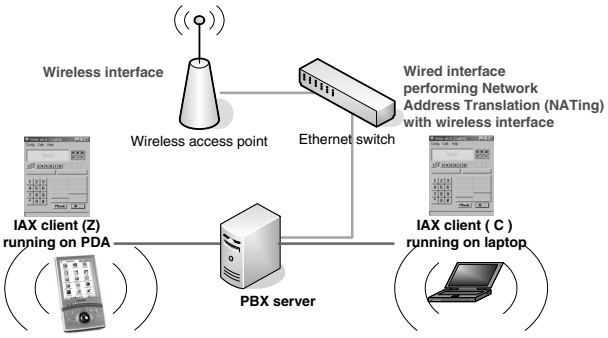

Figure 3: VoIP application set-up

ent trusted configurations of its clients. We also assume that each VoIP client runs on a platform containing SW-TPM. The security scheme for a trusted and secure VoIP registration and communication session proceeds as follows:

Trusted registration of the VoIP client with the PBX server:

- Before any client registers with the server, the platform on which this client is running generates an AIK, and certifies its public portion with respect to its EK, *i.e.*, it obtains an AIK certificate from a Trusted Third Party (TTP). The TTP checks that the AIK has been generated by a valid TPM by using the Endorsement Credential to verify the public part of the EK, and then using the EK to verify the AIK. The TTP then generates an AIK certificate, which is encrypted using a symmetric key S, where S in turn is encrypted using the public portion of the EK. SW-TPM decrypts S, activates and decrypts the AIK credential, and uses it to authenticate itself to a verifier.
- Upon registration, the OS running on the client platform hashes the binary of the VoIP client. The obtained digest is used to extend a specific PCR (PCR 11), which is signed by the generated AIK. The content of PCR 11, corresponding signature, AIK certificate, and platform credentials are sent to the server.
  The server checks the certificates and signature sent, compares PCR 11 to valid VoIP client configurations, and accordingly either registers the client or rejects it.

Once a client has registered with the server, secure VoIP sessions proceed as illustrated in Figure 4 and described below.

Trusted communication between two VoIP clients:

- Client (Z) is launched, and a request is sent to the server for establishing a VoIP session between (Z) and (C). Upon launching client (Z), PCR 11 is reset, and the OS performs a new hashing of the client binary, used to update PCR 11. (This is essential in order to detect any tampering that could have happened between the time of the client registration and the VoIP communication request.)
- If both clients are registered with it, the server sends challenge requests to both (Z) and (C), including a freshness nonce (the nonce is necessary to help prevent replay attacks), request for platform attestation credentials, and integrity measurements.
- Client (C) is launched and PCR 11 is updated using the same method as (Z). SW-TPM of each of the clients signs the value of PCR 11, concatenated with the nonce, using the AIK generated at registration time. Platform credentials, AIK certificate, PCR 11 content, and the corresponding signature are sent to the server.
- The server checks the authenticity of the platform, verifies the signature and its freshness, and AIK credentials, and checks if the PCR values match trusted configurations of VoIP clients. If so, (i) a connection is established between (Z) and (C), and (ii) a 128-bit AES_CBC symmetric session key is generated, encrypted with both (Z)'s and (C)'s AIKs, and sent to them.
- (Z) and (C) decrypt the session key with the private portions of their respective AIKs.
- Once the session key is established, network traffic can be encrypted using the AES_CBC algorithm.

The SW-TPM commands executed are: TPM_MakeIdentity, TPM_ActivateIdentity, TPM_PcrExtend, and TPM_Quote. These SW-TPM commands are executed concurrently on both clients. In addition, SW-TPM decrypts the AIK certificate using its AES_CBC engine. Further overhead is imposed by the integrity measurements of the VoIP clients and the symmetric encryption and decryption of the data traffic between the two clients.

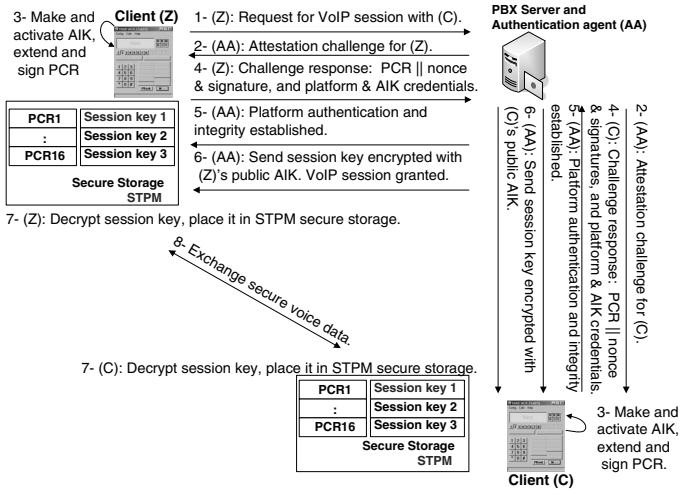

Figure 4: VoIP communication security scheme

In order to identify the overhead imposed by the security scheme, we executed VoIP sessions of varying lengths (one, two, five, and ten minutes) between the PDA and the laptop. In Table 3, rows 5-20, we report the PDA energy and time measurements results as follows: rows 5-12 report measurements for a VoIP session where traffic is protected against eavesdropping attacks only (*i.e.*, data traffic is encrypted before transmission), and rows 13-20 report measurements for a session where traffic is also protected against spoofing attacks (*i.e.*, data traffic is encrypted and hashed before transmission). Figure 5 illustrates the variation of energy consumption and breakup with respect to the VoIP session duration.

4.4 Secure Web Browsing

The Secure Sockets Layer (SSL) is a protocol that provides an authenticated end-to-end communication between a client (*e.g.*, a web browser) and a server (*e.g.*, a web server). It provides mechanisms for mutual authentication of the peers and ensures the confidentiality and integrity of data exchanged between them. SSL, in the context of web browsers, is located between the HyperText Transfer Protocol (HTTP) and Transmission Control Protocol (TCP) layers. SSL consists of multiple protocols, notably: (1) the SSL handshake protocol, which involves a client's authentication of the server, optionally a server's authentication of the client, cipher suite negotiation, and session key exchange, and (2) the Record protocol, which is used to exchange a series of handshake messages and is also responsible for bulk data transfer. It divides the data into manageable blocks, applies the hashing and encryption algorithms established during handshake, adds a header, and transmits the resulting SSL record as a TCP packet.

A security flaw in this protocol is pointed out in [25] – whereas the client can trust a server based on its certificate, it cannot know whether this server is providing the intended service. A solution for tackling this problem is outlined in [7]: (i) binding the SSL private key to the provided application as well, and (ii) storing this bound form in shielded storage, which can be provided by a TPM.

In this work, we use a slightly different procedure to authenticate not only the identity of the server, but also the application that it executes. In addition to sending the server certificate, SW-TPM provided at the server end generates an AIK, obtains an AIK certificate (as described in Section 4.3), and sends it to the client alongside the server certificate. The client checks the AIK certificate as well, and if the certificate

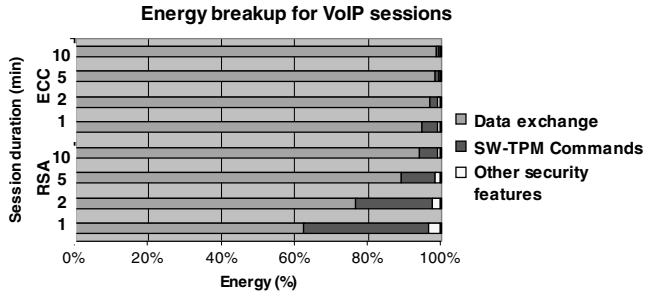

Figure 5: VoIP session energy breakup

is judged trustworthy, it requests a proof of application integrity, *i.e.*, a proof showing that the application that is about to be executed is untampered. This proof consists of integrity measurements of application code (such as security-critical HTML code snippets), signed by the private part of the certified AIK. Once integrity measurements are verified, application data can start to be sent through the SSL Record protocol.

The SW-TPM commands used are: TPM_MakeIdentity, TPM_ActivateIdentity, TPM_PcrExtend, and TPM_Quote (server side). Other overheads are caused by integrity measurements, and the SW-TPM symmetric decryption of the AIK certificate (server side).

Figure 6 shows the secure web browsing scheme we propose, and how the steps required fit in a traditional SSL protocol.

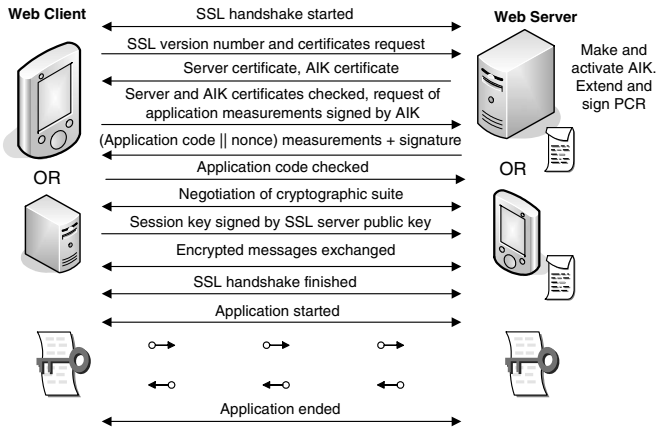

Figure 6: Trusted web browsing scheme

The SSL application running on our handheld device was adapted from OpenSSL, cross-compiled, and patched in order to run on an ARM platform. We consider two cases in which the PDA acts as a client or a server. In each case, we perform server authentication, and server's application authentication in the handshake session using the RSA-AES256-SHA cipher suite, and a 1024-Byte long session ($L = 1024$).

The energy and execution time measurements for the trusted extensions to the web brower are reported in Table 3. Rows 21-22 report results for a trusted SSL session on a PDA acting as a server, and rows 23-24 report results for a trusted SSL session with the client running on the PDA.

5 Conclusion

Trusted computing is a promising paradigm that has applicability to a wide range of embedded systems. For embedded systems in which the cost and size overheads of a separate TPM chip cannot be justified, a sofware-based implementation of TPM functionality (SW-TPM) is an interesting alternative. In this work, we analyzed the execution time and energy overheads for SW-TPM through detailed characterization and measurements on a handheld PDA. In addition to presenting data for individual TPM commands, we also studied the effect of SW-TPM in the context of typical applications. Our results demonstrate that the overheads may be acceptable for some applications, *e.g.*, applications where the TPM functions are used only in the set-up phase, without affecting the dominant data transfer phase. We proposed the use of ECC-based SW-TPM in order to reduce the execution time and energy overheads without compromising security. We believe that this is a practical approach to realizing the benefits of trusted computing in resource-constrained embedded systems.

Table 3: Energy and execution time for trusted applications

Cryptographic algorithm	Untrusted application/ Overall trust overhead		SW-TPM commands overhead	
	Energy(J)	Time(sec.)	Energy(J)	Time(sec.)
Trusted Boot				
RSA	95.542/198.759	48.281/89.495	66.806	28.657
ECC	95.542/137.699	48.281/63.280	5.746	2.442
Secure Storage				
RSA	0.057/75.251	0.023/29.732	75.059	29.661
ECC	0.057/12.026	0.023/4.932	11.823	4.859
VoIP: Voice data encrypted				
RSA	159.243/97.353	60/40.752	88.778	37.376
ECC	159.243/9.446	60/4.213	8.034	3.579
RSA	318.081/98.228	120/41.240	88.778	37.376
ECC	318.081/10.324	120/4.602	8.034	3.579
RSA	804.246/100.867	300/42.356	88.778	37.376
ECC	804.246/12.961	300/6.524	8.034	3.579
RSA	1614.909/105.343	600/44.001	88.778	37.376
ECC	1614.909/17.366	600/7.712	8.034	3.579
VoIP: Voice data encrypted and hashed				
RSA	159.243/98.893	60/40.971	88.778	37.376
ECC	159.243/9.795	60/4.522	8.034	3.579
RSA	318.081/100.121	120/41.508	88.778	37.376
ECC	318.081/11.034	120/4.900	8.034	3.579
RSA	804.246/103.612	300/43.121	88.778	37.376
ECC	804.246/14.708	300/7.237	8.034	3.579
RSA	1614.909/108.879	600/45.630	88.778	37.376
ECC	1614.909/20.861	600/9.191	8.034	3.579
SSL: Server running on PDA				
RSA	0.530/92.455	0.213/38.707	86.271	36.124
ECC	0.530/7.652	0.213/3.387	7.250	3.186
SSL: Client running on PDA				
RSA	0.707/2.449	0.284/1.175	n/a	n/a
ECC	0.707/0.259	0.284/0.124	n/a	n/a

References

[1] "CERT research 2005 annual report." [Online]. Available: http://www.cert.org/archive/pdf/cert_rsch_annual_rpt_2005.pdf

[2] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," *ACM Trans. on Embedded Computing Systems*, vol. 3, pp. 461 – 491, Aug. 2004.

[3] *TCG Glossary*. TCG, 2004. [Online]. Available: https://www.trustedcomputinggroup.org/groups/TCG_Glossary.pdf

[4] "IBM thinkpad." [Online]. Available: http://www.pc.ibm.com/us/thinkpad

[5] "Lifebook S7000 notebook." [Online]. Available: http://www.computers.us.fujitsu.com

[6] "R. Kay, How to implement trusted computing." [Online]. Available: https://www.trustedcomputinggroup.org/news/Industry_Data

[7] S. W. Smith, *Trusted Computing Platforms, Design and Applications*. Springer, 2005.

[8] R. Sandhu and X. Zhang, "Peer-to-peer access control architecture using trusted computing technology," in *Proc. ACM Symp. on Access Control Models and Technologies*, June 2005, pp. 147 – 158.

[9] E. Shi, A. Perrig, and L. V. Doorn, "BIND: A fine-grained attestation service for secure distributed systems," in *Proc. IEEE Symp. Security and Privacy*, May 2005, pp. 154–168.

[10] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A virtual machine-based platform for trusted computing," in *Proc. ACM Symp. Operating Systems Principles*, Oct. 2003, pp. 193–206.

[11] "IBM research report." [Online]. Available: https://www.trustedcomputinggroup.org/news/articles/rc23363.pdf

[12] G. Xu, C. Borcea, and L. Iftode, "Satem: Trusted service code execution across transactions," in *Proc. IEEE Int. Symp. Reliable Distributed Systems*, Oct. 2006.

[13] Trusted Computing Group, "TCG Specification Architecture Overview," Apr. 2004.

[14] "Trusted Mobile Platform." [Online]. Available: http://www.trusted-mobile.org

[15] "Secure extensions to the ARM architecture." [Online]. Available: http://www.arm.com/trustzone

[16] "M. Strasser, TPM Emulator." [Online]. Available: http://developer.berlios.de/projects/tpm-emulator

[17] "Mersenne Twister Random Numbers Generator." [Online]. Available: http://www.math.sci.hiroshima-u.ac.jp/ m-mat/MT/ewhat-is-mt.html

[18] "Sharp Zaurus SL-5600." [Online]. Available: http://www.linuxjournal.com/article/6792

[19] "OpenSSl Project." [Online]. Available: www.openssl.org

[20] D. Hankerson, J. L. Hernandez, and A. Menezes, *Software Implementation of Elliptic Curve Cryptography Over Binary Fields*. Lecture Notes in Computer Science, 2000.

[21] "Sun's elliptic curve technology contribution to the OpenSSL." [Online]. Available: http://research.sun.com/projects/crypto/FrequenlyAskedQuestions.html

[22] "TrouSerS-an open-source TCG Software Stack implementation." [Online]. Available: http://sourceforge.net/projects/trousers

[23] "GRUB TCG Patch to support Trusted Boot." [Online]. Available: trousers.sourceforge.net/grub.html

[24] "Asterisk | the open source PBX." [Online]. Available: http://www.asterisk.org

[25] M. Broekman, *End-To-End Application Security Using Trusted Computing*, 2005. [Online]. Available: http://www.cs.ru.nl/onderwijs/afstudereninfo/scripties/2005/MichielBroekmanScriptie.pdf