A Simple and Efficient Way to Combine Microcontrollers with RSA Cryptography

Echo P. Zhang, Junbin Fang*, Delta C.C. Li, Michael W.H. Ching, T.W. Chim, Lucas C.K. Hui, S.M. Yiu

Abstract—**Microcontroller can be easily adopted in various applications with a variety of peripherals due to its merits of small size, simple architecture and etc. However, the limited computing power restricts its application in cryptography. In this paper, we try to integrate microcontroller with different peripheral devices to support more powerful cryptography computation in a simple and efficient way. Based on the most popular open source microcontroller development platform, Arduino, we design and develop a cryptographic hardware device for a real-life application which provides data protection functions for authority and integrity with RSA cryptography supported. With the peripherals Java card, our Arduino-cored solution is able to efficiently generate digital signature of photos taken by smart phone using the asymmetric cryptographic algorithm, RSA, which has a poor performance if it is directly implemented on microcontroller. The experimental results show that the device can finish a RSA 1024-bit encryption in 82.2 microseconds, which is reasonable in real application scenario and illustrates the feasibility of implementing more complicated cryptographic system using microcontroller.**

Index Terms—**microcontroller, Arduino, cryptography, Java card, RSA.**

I. Introduction

In the recent decade, microcontroller is getting popular application in portable devices, embedded system and mobile platform due to its integrated architecture, rich functionalities and increasing processing power. Arduino is one kind of microcontroller with open source platform. It has been widely adopted in different areas recently. Both Google and Apple have decided to use Arduino as the accessories for their products. Another outstanding feature of Arduino is that it is easy and convenient to be connected to different peripheral devices such as SD card reader, smart card reader, Bluetooth adapter and etc. In short, Arduino has a broad developing future because of its low cost, cross-OS scalability, open source and easy usage features. As a result, various multi-functional applications can be developed on this platform.

Consider a real-life application. *In the collection of digital evidence, how can a picture taken by a smart phone be properly verified and be presentable as from a trusted source?* To achieve this goal, digital signature should be used. However, the signature should not be generated by the smart phone itself as one can use some Apps to forge signature easily. Therefore, a convenient hardware module (assumed to be trusted) is desired to complete this procedure with high efficiency and low-power consumption. For this purpose, Arduino should be the best choice to be the core of the hardware solution with various peripherals.

According to the existing literature, block cipher performance on Arduino's microcontroller is good [1]. However, due to limitation of the 8-bit micro-processor, any kind of modern cryptographic algorithms, like RSA and ECC, cannot be executed efficiently. In [2], the performance of running RSA and ECC algorithms on Arduino with several different libraries is tested, including AvrCryptolib, Wiselib, TinyECC and Relic-toolkit is tested. As for AvrCryptolic, the running time is 25.0s for RSA-512bit and 199.0s for RSA-1024bit algorithms. Normally the shortest key length for commercial application of RSA is 1024 bits. This clearly indicates that using the microcontroller circuit to provide security RSA cryptography (i.e. at least 1024 bits) is not practical.

One direction to solve this problem is to use special ECC algorithms (Elliptic Curve Cryptography) to replace RSA, since it is believed that ECC will provide the same security level as RSA, with a shorter key length. From the experimental result of [2], we find that the ECC outperforms RSA on Arduino with the involvement of hardware. However, to have such a hardware-based ECC implementation, additional RAM needs to be added to the Arduino, which is a precious resource in the microcontroller. Also, using ECC will create an extra software compatibility problem. If an IT system is built in which the microcontroller with that ECC algorithm is used, the other software components need to add a special ECC cryptographic library to work with the microcontrollers. Therefore, it is still better if there is a solution that enables the microcontrollers to use standard RSA (i.e. those already being used in the X.509 PKI standard), to avoid the microcontroller memory problem and the software compatibility problem.

A. Contribution of This Paper

In this paper, we try to explore the possibility of integrating different peripheral devices with Arduino platform to protect the authority and integrity of data in a simple and efficient way. The major contribution of our work is that we utilize the peripheral device, the JavaCard, to enhance the performance of RSA in our proposed Arduino system. JavaCard, being one form of SmartCard, has an efficient hardware circuit for modern cryptography including RSA-1024. Therefore our solution is to find an efficient way to enable the microcontroller to call the SmartCard to carry out RSA cryptography. Note that although SmartCard is a mature technology that the driver from personal computer is available, such driver for microcontroller does not exist. Also a full implementation of SmartCard driver is too much burden for the microcontroller. Therefore, our solution is to implement a trim-down version of SmartCard driver, enough

Manuscript received July 14, 2013.

Echo P. Zhang, Junbin Fang (corresponding author, e-mail: junbinfang@gmail.com), Delta C.C. Li, Michael W.H. Ching, T.W. Chim, Lucas C.K. Hui and S.M. Yiu are with the Department of Computer Science, The University of Hong Kong, HKSAR, China.

Junbin Fang is also with the Key Laboratory of Optoelectronic Information and Sensing Technologies of Guangdong Higher Education Institutes and the Department of Optoelectronic Engineering, Jinan University, Guangzhou, China.

for driving the JavaCard to carry out RSA cryptography, and lightweight enough for the microcontroller to support. As a summary, by linking up the JavaCard, we enable the microcontroller to carry out RSA cryptography efficiently.

The rest of this paper is organized as follows. In Section II, we present the framework of our integrated hardware platform with the technical details of integrating Arduino platform with the peripheral modules, e.g., Java Card module. Our data protection solution for the real-life application of digital evidence is introduced in Section III. The performance of our solution is discussed in Section IV. Section V concludes this paper.

II. The Integrated Hardware Platform and Modules

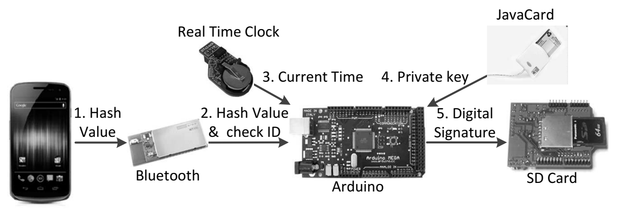

Fig. 1: The Components of Integrated Hardware Platform

The integrated hardware device is shown in Fig. 1. The major components in the integration include:

- Arduino - which is the heart of our integration;
- Java Card - which can be considered as a cryptographic component for providing cryptographic functions;
- Bluetooth adapter - which facilitates wireless communications with an Android phone nearby;
- Real Time Clock (RTC) - which is battery-driven and a standalone component for retrieving the current time;
- SD Card - which provides storage capabilities.

Each component has its unique features and some complicated tuning is required before it can be integrated with Arduino.

A. Java Card Module and Arduino

Here, we utilize Java card to assist Arduino implementing RSA. Previous research works have shown that Java card is a self-contained cryptographic platform with proper key management. In our implementation, we load different Java card Applets (or known as cardlets) onto the Java card together with symmetric or asymmetric keys. We then use the Arduino board to invoke cardlets to execute message digest and symmetric or asymmetric cryptographic functions. With reference to the CheckPoint Charlie project of Philips ARM Design Contest [3], the circuit shown in Fig. 2 was designed to connect a contact type smart card under ISO-7816 standard [4] with an Arduino board.

For the connection of Arduino and Java card, there are three main points needed to be paid more attentions.

1) Serial Communications: The first point is serial communications. In ISO-7816 standard, serial communications with a contact type smart card requires even parity bit and 2 stop bits for every 8 bits of data transfer, which is different from the default setting of Arduino UART interface with no parity and 1 stop bit.

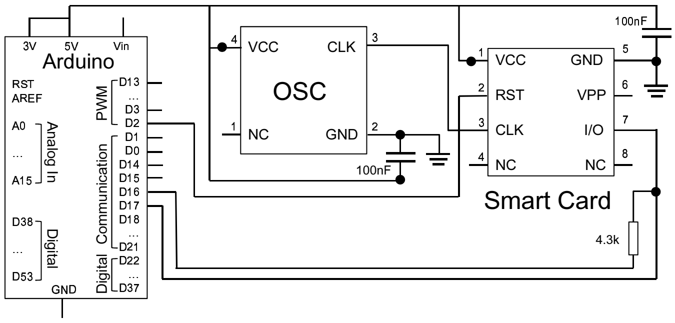

Fig. 2: Schematic of JavaCard Module

Only a single I/O pin is available on a contact type smart card (Pin 7) for asynchronous half-duplex mode communication. To connect the card's serial I/O pin with Arduino UART interface, a 4.3K resistor is installed on the transmission line to avoid message loop back. Packet sequence control and standard timeframe are also important for the implementation.

2) Power Cycle: The second point is the power cycle. To power up and reset a contact type smart card to the initial operation state with Arduino, the following conditions are required:

(1) RST is in state Low
(2) VCC shall be powered
(3) I/O in the Arduino shall be put in reception mode (by controlling RX pin register)
(4) VPP shall be raised to idle state (applicable to smart card using VPP for programming)
(5) CLK shall be provided with a suitable and stable clock
(6) RST is in state High after 40,000 clock cycles

Answer to Reset message is expected from the smart card, which contains specification and mode of operation supported by the smart card.

3) Protocol Parameter Selection and Baud Rate: The last point is protocol parameter selection and baud rate. Serial baud rate of the contact type smart card is controlled by the clock signal of an external oscillator. With a 3.579545MHz clock signal, the baud rate is about 9,600 bps. However, a baud rate of 9,600 bps is obviously slow when compared to the processing speed of microcontroller and smart card processor nowadays. If the smart card supports Protocol and Parameter Selection (PPS) function, it is possible to adjust the baud rate before the actual smart card APDU operation starts.

Two parameters, F (clock-rate conversion factor) and D (bit-rate conversion factor), can be adjusted as follow:

$$CurrentETU = F(D * f)$$

$$Baudrate = 1/CurrentETU$$

In general, the PPS bytes contain 4-7 bytes of data describing the mode of communication required by the host device. Assume F = 372, D = 12, and the standard baud rate is multiplied by 12 times, the PPS bytes in HEX are 0xFF1118F6. The corresponding binary format are:

PPSS: B11111111 - PPS request
PPS0: B00010001 - PPS1 present, T=1
PPS1: B00011000 - TA1 where F=372, D=12

PCK: B11110110 - XOR all bytes to zero

The operation sequence of activating a Java card with PPS on the Arduino board is as follows::

(1) Start serial communication at 9600bps.
(2) Set serial transmission mode as 8-bit data, even parity, 2 stop bits.
(3) Reset the JavaCard.
(4) Wait for 4-400 ETU cycles.
(5) Receive the ATR message form the JavaCard.
(6) Check if PPS is supported, and process other card information from ATR.
(7) If PPS is supported, send the PPS byte 0xFF1118F6.
(8) If no error occurs, the JavaCard echo back the PPS.
(9) Terminate the serial communication.
(10) Restart the serial port at 115200 baud rate.
(11) Continue APDU operation at current baud rate.

B. *Bluetooth Module and Arduino*

Another important module in our design is the bluetooth. To enable wireless communication with the Arduino board, we have developed a Bluetooth module that can be attached to UART serial interface of Arduino with Bluegiga WT11 Bluetooth module [5]. The module is composed of a 5V to 3.3V voltage regulator and LEDs indicating power status and connection status.

Using the WT11 module, Arduino can interact with different types of Bluetooth devices using various Bluetooth profiles supported. Such flexibility can help us achieving different security requirements over the wireless communication channel.

There are also three points needed to draw attention when connect bluetooth with Arduino.

1) Device Paring: The first point is device paring. As the Arduino is not attached with human interface device by default, auto pairing of bluetooth device is preferred. The default auto pairing mechanism of WT11 module makes use of a four-digit PIN. Any connecting device presenting the correct PIN can be paired. Yaniv Shaked and Avishai Wool [6] suggested that the short PIN can be cracked using wireless packet eavesdropping technique and re-authentication attempt attack. One the PIN is compromised, intrusion detection becomes infeasible and the system is subjected to vulnerability over the wireless communication channel.

To address this issue, additional logic is added to the Arduino to handle pairing request to the WT11 module. Following protocol can be used for auto pairing using bluetooth device address filtering:

(1) A Bluetooth device calls WT11 module over RFCOMM layer for connection;
(2) If a correct 256-bit link key is presented by the connecting device, pairing is not required;
(3) Else WT11 module sends 40-bit Bluetooth Device Address to Arduino;
(4) Arduino verifies the address with records on external storage / over the network;
(5) If the address can be verified, Arduino responses WT11 module to accept the pairing. A new 256-bit link key is stored on WT11 module with the address;
(6) Else Arduino responses WT11 module to drop the connection.

The Arduino can control the wireless access by whitelisting / blacklisting. This can prevent intrusion of the system by PIN cracking. However, the level of security measure using the 256-bit link key after the pairing phase is not affected by this protocol, and the link level security is bounded by the bluetooth standard.

2) Data Communication: Once the pairing of connecting device and the WT11 module is completed, RFCOMM connection can be established. WT11 module can notify Arduino the device address of incoming ring and enter data mode automatically. The programmable I/O pin (PIO7) connecting an LED is set to high when the connection is up, and this pin can also be connected to a digital input pin of Arduino as interrupt.

The data rate of Arduino UART interface is limited by the processor speed and the 128 bytes serial buffer. A typical receive loop of serial data can be defined as follow in the Arduino:

```
while(Serial.available()){
    byte inByte = Serial.read();
    // process the incoming byte...
}
```

At high serial baud rate, Arduino should process the incoming bytes as quick as possible to prevent buffer overflow. In case of buffer overflow, Serial.read() function returns garbage byte in the loop. However, if link quality drops and buffer underflows at the receiver side, which simply breaks the $Serial.available()$ loop. Both cases suggest that flow control and error control are required. We have tested the following protocol over the bluetooth connection:

- Pack the raw message into 1024-byte packets;
- Sender sends a packet with sequence number;
- Receiver expects a 1024-byte packet and responses ACK message;
- Sender sends next packet if ACK is received;
- The transmission is completed by escape sequence number.

The error handling mechanism (in case of no ACK response) should be agreed by both sender and receiver with proper timeout handling, otherwise the receiver loop may keep running forever. Interrupt driven timeout is not directly supported by the Arduino environment and should be implemented using Interrupt Service Routine related libraries.

With the above flow control, the achievable data rate is around 80% of the max data rate at 115200 baud. Predefined OPCODE (such as “SIGN” and “VERIFY”) are used in our implementation to limit the functions of the system exposed to other bluetooth devices. If the connected device does not follow the flow with predefined OPCODE during the data exchange, Arduino can switch the WT11 module from data mode to control mode and disconnect the device, which can prevent the system from buffer overflow attack.

3) Serial Port Profile: The Serial Port Profile (SPP) emulates a serial port connection between two bluetooth devices over the RFCOMM layer, which provides an abstraction of those calling and answering processes. bluetooth device supporting SPP, such as Android mobile platform, can establish a serial port connection over bluetooth directly, and the underlying processes such as pairing and ringing are triggered automatically. From the application point of view, data communication is over the ad-hoc serial port connection established.

If multiple bluetooth hosts are required to be connected at the same time, other network profile such as Personal Area Networking Profile (PAN) should be loaded to the WT11 module.

C. Real Time Clock Module and Arduino

Timestamp is an important kind of information for designing and implementing a security protocol to prevent man-in-the-middle attack and replay attack. Timestamping also can associate objective evidence to the data being protected.

Implementation of timestamp with Arduino depends on the power state of the board. Clock cycle count can be returned by $mills()$ or $micros()$ function call, but the counter is reset after a restart of the board. To provide a more reliable timestamp for the security application, an external real time clock module running on battery is connected to the Arduino. The real time clock module consists of a DS1307 clock, a crystal (connected to $X1$ and $X2$) for oscillator and a battery (connected to Vbat) for running the clock under battery-backup mode. A standard alone DS1307 clock can run on battery for a few years at 500nA[7].

I2C interface is used to connect the Arduino and the real time clock module. Developer can make use of $< Wire.h >$ library to handle the communication over I2C interface. Following protocol should be used in getting / setting the time value of the module:

(1) Define I2C address of the connected module (e.g. 0x68)
(2) Begin transmission on the I2C address
(3) Transmit a zero byte
(4) Transmit or request 7 bytes representing the time value data
(5) End transmission

The real time clock module can provide seconds, minutes, hours, day, date, month, and year information with leap-year compensation.

D. SD Card Module and Arduino

With limited amount of memory available on the microcontroller of Arduino, external storage is required for dumping output of the running application. Even if the output data can be transmitted over a network connection, temporary non-volatile storage plays an important role in case of network or power failure. The external storage is also useful for data logging or storing routing path that changes from time to time. Arduino can be connected to a SD flash memory card with the SPI bus. Both FAT16/FAT32 file format on SD card are supported.

1) Device Connection: The SD card is running in SPI mode when connected with the Arduino. Arduino can use the Slave Select(SS) pin to select the target device over the SPI bus. As the master host, developer can define multiple SS pins on Arduino and attach multiple slave devices to the bus. Different types of slave devices (flash memory, network interface, etc.) can be attached at the same time. The serial communication is under synchronous full duplex mode over the MISO and MOSI pins. Throughput of SPI bus is higher than other interfaces provided the Arduino.

2) SD Card Library: Using the sdfatlib developed by William Greiman [8] and the $< SD.h >$ wrapper of SparkFun [9], accessing the SD card storage is convenient. Developer can first list out the directories and files on the SD card, and open a file by the full file path. File should be open in either FILE_READ or FILE_WRITE. Under FILE_READ mode, content of open file can be accessed from start to EOF. Under FILE_WRITE mode, data can be written to a new file or appended to an existing file. File point operation is similar to C file reading / writing, and the operation is byte oriented.

3) Data Format: Our implementation uses the SD card to store the cipher data of the security application. Digital digest and PKI signature on input data are usually in binary format. Although the binary data can be stored directly onto the SD card, such format is not human readable and not portable to other application if the stored data is dumped out directly. So we use the $sprintf()$ function to convert binary data into HEX string representation for storage, and $sscanf()$ is used for forming binary array of data from the HEX string stored. The HEX string comes in standard length with respect to the cryptographic function being used, and standardize the exchange format for cipher data outside the hardware of our own application.

III. Application

A. Registration

There are two phases for the application which is dispicted in the introduction: Registration and Verification. During the phase of Registration, a customized application is needed on the customer's Android mobile phone. We develop this customized application to carry out the simple behavior of taking a photo and gathering metadata such as current time, GPS location, etc. Then the application generates a hash value on these information together with the binary data of the image,

$$\mathsf{Hash} = \mathsf{SHA1}(\mathsf{Image}|\mathsf{Metadata})$$

After receiving the hash value from the mobile phone, the Arduino collects the following information from various components: (1) The phone's identity from its Bluetooth adapter and (2) the current time from a Real Time Clock. With these information, the Arduino requests the Java card to generate a digital signature. The signature is generated as follows:

$$\mathsf{Sign} = \mathsf{RSA}_{\mathsf{Enc}}(\mathsf{Priv}_{\mathsf{Arduino}}, \mathsf{Hash}_{\mathsf{Phone}}|\mathsf{Time}|\mathsf{ID}_{\mathsf{Phone}})$$

where Priv_Ard refers to the Arduino's private key as stored on the Java card (retrieved via the Java card reader). The digital signature, together with the time and device ID are then stored in the SD Card for future verification purpose.

B. Verification

As for the Verification phase, for digital signature verification purpose, the user selects a stored photo and the related metadata. The Arduino looks up from the SD Card the hash value and performs a digital signature to verify that the hash was submitted by a phone with the same ID at the time specified by the stored value. The decryption is done using the public key of the Arduino as stored in the JavaCard.

TABLE I: Running Time of Different Algorithms in Our Device and PC

	Arduino + JavaCard	JCOP on Windows
AES 128bit Encrypt	62.2ms	46.459ms
AES 128bit Decrypt	59.3ms	49.0299ms
RSA 1024bit Encrypt	82.2ms	74.6552ms
RSA 1024bit Decrypt	357.9ms	345.7577ms
SHA1 4 Blocks	60.1ms	52.9774ms

Such a verification process validates the phone-time-image tuple, and allows the usage of this information for official evidence as long as the digital signature is performed using a key signed by a trusted authority.

IV. Performance

The performance of this integrated device can be affected by many different aspects.

A. UART

First of all is UART. Theoretically, the baud rate of UART serial interface is limited by the accuracy of clock signal supplied to the channel. Some of the peripheral devices, such as Bluetooth module, can achieve higher baud rate up to 3 Mbps. However, if different serial devices are running at different baud rates, additional waiting cycles induced by $delay()$ function are required, which in turn may create unnecessary overhead and I/O error in the Arduino. Our integrated solution unifies the maximum UART baud rate to 115,200 bps among all serial interfaces. Achievable data rate then becomes about 80% (about 100 kbps) after applying the flow control protocol over the application.

B. Cryptographic Functions using Java Card

The second aspect affected the performance is cryptographic functions using Java card. We load all different cryptographic functions onto a JCOP21 v2.3.1 JavaCard. The data block size was set to 128 bits. We then perform a series of tests. We compare the results with directly invoking JCOP operations on a Java card from a Windows 7 x64 computer environment via a USB 2.0 card reader. The results are shown in I.

From the results, it can be shown that the performance of driving a Java card over Arduino is comparable to the performance of standard JCOP console on Windows computer. The only difference is caused by overhead in the transmission protocol which is not well optimized in our implementation.

C. Bus Speed between Arduino and SD Card

SPI bus speed of Arduino is at 500 kbps, and most of the SD flash memory card can achieve a few Mbps data rate. Whereas I/O throughput is not a major concern in our implementation. With external flash memory, the bus speed is relatively slow when compared to the single cycle read and write operations on internal SRAM. If intensive table lookup operations are required, the lookup data should still be placed in the internal flash memory or loaded into the SRAM, rather than from the external SD card.

D. Real Time Clock

The real time clock retrieves 8 bytes of timestamp data, including the first zero byte being transmitted. Such an operation takes around 1 ms. On the other hand, converting binary-coded decimal timestamp into unsigned integer takes around 2.3 ms. Thus the overall operation of timestamping can be completed in 4ms.

V. Conclusion

In this paper, the use of hardware platform for building a real-life application has proved to be satisfactory and promising in terms of speed, versatility and security. As one of the most common micro-processor, Arduino can be utilized in the application to provide various security features. However, due to Arduino's limited processing power and memory, Arduino yields a pretty poor performance especially when RSA algorithms are involved. Our research work utilized the good compatibility of Arduino. By integrating Java card, we accelerate the efficiency of RSA encryption and decryption computation on Arduino. Besides, we also connect Arduino with Bluetooth adapter, real-time clock and SD card module to complete an interesting and useful application for Android smart phone.

References

[1] M. Gerd, "Encryption with Arduino AES-256 and RSA-512," http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1276073358/, 2012, [Online; accessed 19-July-2012].
[2] J. Arkko, H.-M. Rissanen, A. Keranen, and M. Sethi, "Practical considerations and implementation experiences in securing smart object networks," 2012.
[3] C. Cossio, "The checkpoint charlie," in *Philips ARM Design Contest 2005, Spain*, 2005.
[4] "ISO/IEC 7816 Identification cards - Integrated circuit cards," 2005.
[5] Bluegiga, "Wt11 data sheet," http://www.iearobotics.com/personal/ricardo/proyectos/skybluetooth/docs/wt11.pdf, 2007.
[6] Y. Shaked and A. Wool, "Cracking the bluetooth pin," in *Proceedings of the 3rd international conference on Mobile systems, applications, and services*. ACM, 2005, pp. 39–50.
[7] D. Semiconductor, "Ds1307 64x8 serial-real time clock," 2008.
[8] W. Greiman, "sdfatlib under gnu gpl v3 license," http://code.google.com/p/sdfatlib/, 2012.
[9] S. Electronics, "Sd under gnu gpl v3 license," http://arduino.cc/en/Reference/SD/, 2010.