IEEE CS Security and Privacy Workshops

The BlueJay Ultra-Lightweight Hybrid Cryptosystem

Markku-Juhani O. Saarinen
REVERE SECURITY
4500 Westgrove Drive, Suite 300
Addison, TX 75001, USA.
mjos@reveresecurity.com

***Abstract*—We report on the development of BlueJay, a hybrid Rabin-based public key encryption cryptosystem that is suitable for ultra-lightweight (total 2000-3000 GE) platforms such as microsensors and RFID authentication tags. The design is related to authors' Passerine and the Oren-Feldhofer WIPR proposals, but is suitable to a wider array of applications. The encryption mechanism is significantly faster and the implementation more lightweight than RSA (even with public exponent 3) and ECC with the same security level. Hardware implementations of the asymmetric encryption component of the hybrid cryptosystem require less than a thousand gate equivalents in addition to the memory storage required for the payload and public key data. An inexpensive, milliscale MCU SoC BlueJay implementation is reported and compared to RSA-AES on the same platform. The private key operation (not performed by the light-weight device but by the sensor network base station or a data acquisition reader) has roughly the same complexity as the RSA private key operation.**

I. INTRODUCTION

Most modern hand-held and portable devices such as smart phones and tablets are able to support standard cryptographic algorithms of arbitrary strength. However, there are emerging application areas where public key cryptography is not seen as a viable option [1], [2].

- Wireless sensors in military and health care have shrunk during the past two decades from inch-scale to millimeter-scale [3]–[6]. These are often used to transmit sensitive information and hence have a requirement for message confidentiality, integrity and sender authentication.
- A passive RFID tag or a contactless smart card (such as an e-passport) draws its operating power from a reader and has to have a relatively low individual manufacturing cost. These applications require at least uncloneability and privacy (anonymity).

A. Where Public Key Encryption is Enough

There are clear advantages in using public key cryptography instead of secret-key cryptography in the ultra-lightweight domain. It is often sufficient for the lightweight party to only implement public-key encryption without decryption (private key) functionality.

1) Sensor Fields: If a multitude of sensors in a sensor field share a "network key", a compromise of a single node and thereby the shared key may put the whole network out of action. If the purpose of that sensor field is to transmit sensory information to data collector node(s) in a secure fashion, it is sufficient to encapsulate that data using the collector's public key at source.

2) Authentication (RFID): Another application area that may be used (in conjunction with the first) is in public key authentication protocols. For an overview of applicable public key protocols, see [7].

3) Secure Logging: If an embedded device is to log sensitive information in tamper-resistant "one-way" fashion into a storage device such as flash memory, a lightweight public key algorithm may be used. After the public-key encrypted data has been written to storage, even the device itself cannot access the plaintext data.

II. BLUEJAY IS A PASSERINE BIRD

We use the name Passerine [8] to refer to an implementation of the Rabin public key encryption algorithm [9] utilizing Shamir's randomized multiplication [10], [11] technique, payload encoding into the randomizer and the Chinese Remainder Theorem (CRT) to make the public key encryption operation more efficient [12].

Passerine differs from comparable light-weight Rabin public key encryption systems such as the Oren–Feldhofer WIPR [13]–[15] or Shamir's original scheme [10] by two main novel distinguishing factors:

1) Our implementation fully utilizes the available message space by encoding payload data into the randomization mask.
2) All arithmetic in the encryption operation is performed using CRT techniques modulo a set of small, register-sized numbers. There is no need for big integer arithmetic.

As a result, the Passerine cryptosystems have superior encryption performance to both RSA and ECC systems. The decryption (private key) operation is roughly as demanding as the private key operation of the RSA algorithm.

DOI 10.1109/SPW.2012.11

A. The classical Rabin cryptosystem

The well-known Rabin cryptosystem derives its security from integer factorization. Let $n = pq$ be the public modulus of sufficient size to resist attacks via factorization. The large primes p and q constitute the secret key. To encrypt a plaintext block $0 < X < n$ in the Rabin cryptosystem, it is simply squared modulo n to produce a ciphertext block C.

$$C = X^2 \pmod{n}. \tag{1}$$

To decrypt, the square root of C can be efficiently computed only if the secret factors p and q of n are known. In fact Rabin showed that recovering entire X from C and n alone was equivalent to factoring n. No such direct equivalence proof is known for RSA.

There are up to four square roots for each C; special encoding must be used to guarantee that the correct plaintext is identified. The square root operation is roughly equivalent to the RSA private key operation in complexity.

B. Randomized multiplication

The modular reduction step of Equation 1 is more computationally demanding than the squaring operation itself. To overcome this, randomized multiplication may be used. For each encryption operation we select a non-repeating random masking quantity $Y > n$ and transmit

$$C = X^2 + Yn. \tag{2}$$

It can be shown that this scheme is as secure as the original Rabin system, while the modular reduction step has been replaced with a cheaper multiplication operation [10]. The decryption operation is the same as in the original scheme. The downside of Shamir's original variant is that the ciphertext is twice as long as the plaintext.

C. Encoding payload in the randomization mask

The mask Y can be recovered if the private key operation to solve X is successful:

$$Y = \frac{C - X^2}{n} \tag{3}$$

We generate the Y mask by using a light-weight symmetric authenticated encryption algorithm such as Hummingbird-2 [16]. Its unique random 128-bit symmetric session key is stored in the block X.

The symmetric authenticated encryption algorithm is also used to encrypt and randomize most of the contents of X (apart from the session key) and to produce a secure authentication tag for the whole message. Hence the ciphertext has the same length as the plaintext, secret key and authentication tag combined.

Note that after the secret symmetric session key is recovered using the private key operation, it can be used to decrypt an arbitrary amount of further data with entirely symmetrical operations. Only the random session key needs to be protected in a fully asymmetric fashion.

D. Use of a residue number system

A key difference between Equations 1 and 2 is that the randomized variant 2 can be executed without division and therefore a residue number system can be used.

For an arbitrary integer b, one may compute $C \bmod b$ by reducing the message quantities X, Y, and the public modulus n modulo b and performing the arithmetic operations in $\mathbb{Z}_b$, the ring of integers modulo b:

$$C \equiv X^2 + Yn \pmod{b}. \tag{4}$$

In BlueJay, the "base" super-ring $\mathbb{Z}_B$, $B > C$, consists of small word-sized factors $B = b_0 b_1 b_2 \cdots$. Arithmetic operations in each subring $\mathbb{Z}_{b_i}$ may be performed independently from each other. We have a system of simultaneous linear congruences:

$$\begin{aligned} C_0 &\equiv X^2 + Yn \pmod{b_0} \\ C_1 &\equiv X^2 + Yn \pmod{b_1} \\ &\cdots \quad \cdots \quad \cdots \end{aligned}$$

Each C_i may be computed using word-sized modular squaring, multiplication and addition operations, without the need for big-integer arithmetic. The word-sized residues C_i are transmitted as an unambiguous representation of ciphertext C. The computationally superior receiving party is responsible for the reconstruction of C from C_i. There are well-known "de-CRT" algorithms; a good introduction to residue number systems and Rabin decryption is given in Sections 2.4, 8.3 and 14.5 of [17].

III. Efficient Ring Selection

From practical perspective it makes sense to choose the base B to consist of co-prime numbers that are close to a word size for easy modular reduction. If the numbers are not co-prime or close to the word size, there will be loss of transmission efficiency. In the following we will consider a case where an appropriate $n \approx 2^{1024}$ is used and the ciphertext size is therefore 2048 bits.

Prime base: Simply choosing the 64 primes that are closest to 2^{32} as the base $B = 4294965793 \times 4294965821 \times \ldots \times 4294967291$ yields a transmission capability of $\log_2(B) \approx 2047.999982$ entropy bits, which is very close to the channel optimum. However, the storage requirement for those 64 primes alone makes this selection inappropriate for many applications.

Linear sequence: To save implementation space, one may consider using linear sequences as the base. We can illustrate this by choosing $b_i = 2^{32} - 1407 - 9699690i$. With this sequence of base words we can decode numbers up to $C < \prod_{i=0}^{63} b_i \approx 2^{2026.7571}$ uniquely with 64 words. The transmission rate is roughly 98.96% in this case. This sequence works nicely because the "slope constant" 9699690 is the product of the eight smallest primes and therefore ensures that the numbers sequence are not divisible by those (since $\gcd(b_0, 9699690) = 1$).

Table I
BASE COMPRESSION BITS OF THE REFERENCE IMPLEMENTATION. THESE ARE USED TO QUICKLY GENERATE 65 COPRIME "BASE" NUMBERS.

$$\Delta_{1\ldots 64} = (\begin{array}{cccccccc} 0, & 0, & 0, & 0, & 1, & 1, & 1, & 0, \\ 1, & 1, & 0, & 0, & 1, & 0, & 1, & 0, \\ 0, & 0, & 1, & 0, & 0, & 1, & 1, & 0, \\ 0, & 1, & 1, & 1, & 1, & 0, & 0, & 0, \\ 1, & 0, & 0, & 1, & 0, & 1, & 0, & 0, \\ 0, & 0, & 0, & 0, & 0, & 1, & 0, & 1, \\ 0, & 1, & 1, & 0, & 0, & 1, & 1, & 0, \\ 1, & 0, & 0, & 0, & 1, & 1, & 0, & 0 \end{array}).$$

Base compression: We have discovered that by setting $b_0 = 2^{32} - 3$ and computing each subsequent b_i as

$$b_{i+1} = b_i - \Delta \text{ where } \Delta \in \{90, 120\}, \tag{5}$$

we can get to near-optimal capacity of $\log_2 \prod_{i=1}^{64} b_i \approx 2047.999664$ bits or 99.999984%. Each Δ index can be stored as a single bit.

Table I gives the delta bits we use in our reference implementation. The table gives Δ_i values that can be used to generate a descending sequence of 65 coprime numbers $b_0, b_1, \ldots b_{64} = 4294967293, 4294967203, \cdots, 4294960753$. Start with $b_0 = 2^{32} - 3$ and compute $b_i = b_{i-1} - 30\Delta_i - 90$.

We have discovered similar "two-step sequences" for up to 4096 bit capacity with a 32-bit word size, but with larger Δ values.

Note that since each one of the b_i is odd, $b = 2^{32}$ can also be used. In this special case the algorithmic modular reduction step is not necessary as the least significant 32 bits of X, Y and n may be used directly.

IV. A BIT-SERIAL PASSERINE IMPLEMENTATION WITH SMALL REGISTERS

BlueJay is designed to allow straightforward bit-serial hardware implementation using small registers for the public key encryption function. The more complicated decryption function can not be implemented this way.

Let $\mathsf{red}(r, b)$ denote conditional subtraction of a modulus b from r, $0 < r < 2b$:

$$\mathsf{red}(r, b) = r \bmod b = \begin{cases} r & \text{if } x < b \\ r - b & \text{if } r \geq b \end{cases}$$

If we have $b = 2^w - \delta$ for some word size w and relatively small δ, we may implement reduction with overflow bits. This may be more efficient as it allows us to process more than one bit at a time:

$$\mathsf{red}'(r, b) = (r \bmod 2^w) + \delta \left\lfloor \frac{r}{2^w} \right\rfloor. \tag{6}$$

In Equation 6, the overflow bits are multiplied by δ and added to r truncated to w bits. This works correctly since $2^w \equiv \delta \pmod{b}$.

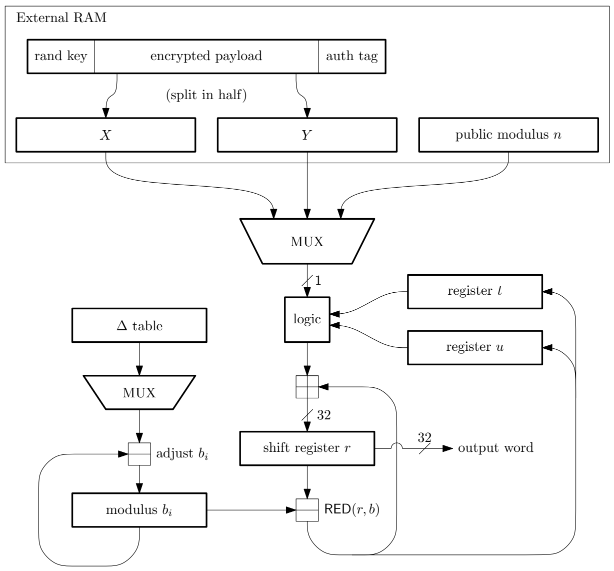

Figure 1. Simplified block diagram of a bit-serial BlueJay hardware implementation. Ciphertext $C_i = C \bmod b_i$ is produced and transmitted one word at a time.

Algorithm 1 gives the complete reduction and algebraic step for producing a single word of asymmetrically encrypted ciphertext. We see that this type of hardware implementation requires only a single word-sized (32-bit) shift register with conditional subtraction $r \leftarrow \mathsf{red}(r, b)$, two registers for storage and $\lceil \log_2 bXY \rceil$ clock cycles.

Figure 1 illustrates the operation of Algorithm 1 and the components required in the operation of this algorithm.

Note that when implementing these algorithms, the "if" statements should have "else" counterparts that implement non-operations with equivalent time and power consumption profile to counter side-channel attacks [18], [19].

V. THE BLUEJAY HYBRID CRYPTOSYSTEM

BlueJay is a combination of the Hummingbird-2 lightweight authenticated encryption algorithm and Passerine optimized for a 1024-bit public modulus n and 32-bit register size.

Since Hummingbird-2 requires only 2159 Gate Equivalents (GE) [16] (and can be as small as 1200 for encryption only), the total hardware size required to implement BlueJay is under 3000 GE, in addition to the memory used to store payload and public key information.

Algorithm 2 describes how Hummingbird-2 is used with Passerine in BlueJay. Here the parameters are $w = 32$ and the loop on line 6 has 65 iterations. One ciphertext block can carry 1792 bits of payload data, a 128-bit secret key and a 128-bit authenticator. The same 128-bit key can be used to encrypt further data in entirely symmetric fashion.

Algorithm 1 $\mathsf{psrn}(x, y, n_b, b)$: Produce one asymmetric ciphertext word from plaintext $(x_{\lceil \log_2 X \rceil} \cdots x_1 x_0 \mid\mid y_{\lceil \log_2 Y \rceil} \cdots y_1 y_0)$, with binary representations $X = \sum_{i=0} 2^i x_i$, $Y = \sum_{i=0} 2^i y_i$, given the precomputed quantity $n_b = n \bmod b$.

```
1:  $r = 0$                                   Clear the $r$ register for computation of $(Yn \bmod b)$.
2:  for $i = \lceil \log_2 Y \rceil, \ldots, 2, 1, 0$ do
3:      $r \leftarrow \mathsf{red}(2r, b)$            Left-shift the contents of $r$ and subtract $b$ in case of overflow.
4:      if $y_i = 1$ then
5:          $r \leftarrow \mathsf{red}(r + n_b, b)$   Add (precomputed) $n_b = n \bmod b$ to multiply that quantity with $Y$.
6:      end if
7:  end for
8:  $t \leftarrow r$                              Store $(Yn \bmod b)$ to register $t$.

9:  $r \leftarrow 0$                              Clear the $r$ register for computation of $(X \bmod b)$.
10: for $i = \lceil \log_2 X \rceil, \ldots, 2, 1, 0$ do
11:     $r \leftarrow \mathsf{red}(2r + x_i, b)$      Left-shift $r$ and append a bit of $x$ to the right, subtract $b$ in case of overflow.
12: end for
13: $u \leftarrow r$                              Store $(X \bmod b)$ to register $u$.

14: $r \leftarrow 0$                              Clear the $r$ register for computation of $(X^2 \bmod b)$
15: for $i = \lceil \log_2 b \rceil, \ldots, 2, 1, 0$ do
16:     $r \leftarrow \mathsf{red}(2r, b)$            Left-shift the contents of $r$ and subtract $b$ in case of overflow.
17:     if $u_i = 1$ then
18:         $r \leftarrow \mathsf{red}(r + u, b)$     Add multiplicand $u = x \bmod b$.
19:     end if
20: end for

21: return $\mathsf{red}(r + t, b)$               Return (and immediately transmit) the ciphertext word $r = (X^2 + Yn) \bmod b$.
```

Table II
RVM0CL v1.02 Cortex M0 crypto library and the BlueJay.

Component	Bytes	Description
`aes128.o`	4430	AES-128 and CBC.
`m0rsa.o`	3980	Raw RSA functionality.
`pkcs.o`	586	PKCS #1 v1.5 encoding and decoding.
`sha1.o`	560	SHA-1.
`drbg.o`	366	Deterministic Random Bit Generator.
`bluejay.o`	292	Passerine (BlueJay).
`hb2.o`	4174	Hummingbird-2 (with MAC).

VI. Comparison on the Cortex M0 Platform

We present results of comparison between RVM0CL [20] (a commercial RSA / AES / SHA library) and BlueJay on a lightweight MCU based on the Cortex M0 / M0+ instruction set [21], [22]. Figure 2 shows NXP LPC1102, a SoC that implements this architecture in an inexpensive, small, and readily available form factor. Note that M0 / M0+ cores (which themselves have an area of typically under 12k gates) do not have a division instruction or hardware, making implementation of public key algorithms based on modular arithmetic especially challenging. Table II gives the implementation sizes of these cryptographic components with the Thumb instruction set of M0.

As expected, BlueJay encryption is significantly faster when compared to RSA. Table III gives a performance comparison of these two algorithms on Cortex M0. The private key operation requires about 180×10^6 Cortex M0 cycles for both RSA and BlueJay. The RVM0CL library requires 812 bytes of working space, whereas BlueJay requires less than 50 bytes if the ciphertext words are transmitted as they are computed.

Table III
BlueJay vs RSA Encryption performance on Cortex M0.

Algorithm	Parameters	Cycles
BlueJay	$n \approx 2^{1024}$	225,000
RSA	$n \approx 2^{1024}$, $e = 3$	929,000
RSA	$n \approx 2^{1024}$, $e = 17$	2,417,000
RSA	$n \approx 2^{1024}$, $e = 65537$	8,297,000

VII. Conclusions

We have described BlueJay, a lightweight hybrid public key encryption algorithm. BlueJay is based on the Rabin cryptosystem and breaking it is provably as hard as factoring. The public key operation is an order of magnitude faster than RSA (and 2-3 orders of magnitude faster than ECC), while the private key operation is almost exactly as demanding as the RSA private key operation. BlueJay is intended for sensor data acquisition, RFID authentication, and secure logging applications, where only the public key operation is required. The algorithm core can be implemented in hardware with less than one thousand gate equivalents or less than 300 bytes of code on the Cortex M0 platform.

Algorithm 2 $\mathsf{bluejay}(p, n)$: Encrypt plaintext p to a recipient whose public key is n.

```
1:  $k \leftarrow$ random 128-bit key
2:  $(z, a) \leftarrow \mathsf{HB2}(p, k)$                  Run Hummingbird-2 on plaintext $p$ with key $k$ to produce ciphertext $z$ and an authentication tag $a$.
3:  $x \leftarrow k \;||\;$ (first half of $z$)         Store 128-bit symmetric key $k$ and $\lceil \log_2 n \rceil - 128$ bits of $z$ in $x$.
4:  $y \leftarrow$ (second half of $z$) $||\; a$        Store $\lceil \log_2 n \rceil - 128$ bits of $z$ and the 128-bit authenticator $a$ in $y$.
5:  $b \leftarrow 2^w$                                  The base modulus is in $b$.
6:  for $i = 1, 2, \ldots, \left\lceil \frac{2 \log_2 n}{w} \right\rceil$ do
7:      $b \leftarrow b - \Delta_i$                     $\Delta_i$ comes from a table lookup; it is compressed as discussed in Section III.
8:      $\mathsf{transmit}\big(\mathsf{psrn}(x, y, n_i, b)\big)$   The quantity $n_i = n \bmod b_i$ either comes from a precomputed table or is reduced on the fly like $x$ and $y$.
9:  end for
10: $\mathsf{transmit}$(rest of $z$)                    Arbitrary amount of symmetric ciphertext.
```

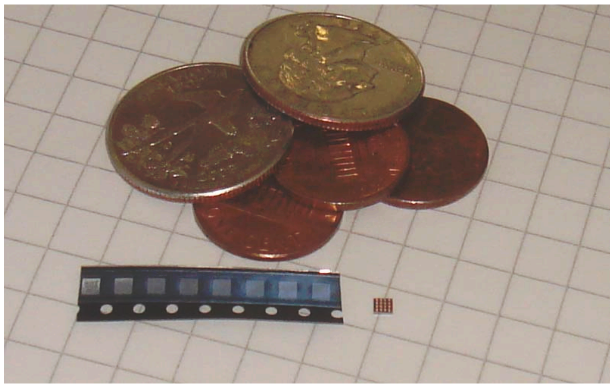

Figure 2. Implementations were compared on the Cortex M0 platform. Pictured is NXP LPC1102, a Cortex M0 self-contained system-on-chip (with an on-chip clock source) that has WLCSP16-packaged external surface area of 2.17 × 2.32 = 5.03 mm^2 or $\frac{1}{128\text{th}}$ of a square inch.

References

[1] G. Gaubatz, J.-P. Kaps, E. Öztürk, and B. Sunar, "State of the art in ultra-low power public key cryptography for wireless sensor networks," pp. 146–150, 2005.

[2] J.-P. Kaps, G. Gaubatz, and B. Sunar, "Cryptography on a speck of dust," *Computer*, vol. 40, no. 2, pp. 38–44, February 2007.

[3] K. Arshak, E. Jafer, and D. McDonagh, "Modelling and simulation of a wireless microsensor data acquisition system using PCM techniques," *Simulation Modelling Practice and Theory*, vol. 15, pp. 764–785, 2007.

[4] B. W. Cook, S. Lanzisera, and K. Pister, "SoC issues for RF smart dust," *Proceedings of the IEEE*, vol. 96, pp. 1177–1196, 2006.

[5] A. Yakovlev, S. Kim, and A. S. Y. Poon, "Implantable biomedical devices: wireless powering and communication," *IEEE Communication Magazine*, 2012, invited paper – to appear.

[6] A. Yakovlev, D. Pivonka, T. H. Meng, and A. S. Y. Poon, "A mm-sized wirelessly powered and remotely controlled locomotive implantable device," in *Proc. IEEE International Solid-State Circuits Conference (ISSCC)*, February 2012.

[7] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*. Springer, 2003.

[8] M.-J. O. Saarinen, "The PASSERINE public key encryption and authentication mechanism," in *NORDSEC 2010*, ser. Lecture Notes in Computer Science, T. Aura, K. Järvinen, and K. Nyberg, Eds., vol. 7127. Springer, 2011, pp. 283–288.

[9] M. C. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," Massachusetts Institute of Technology / LCS, Tech. Rep. 212, 1979.

[10] A. Shamir, "Memory efficient variants of public-key schemes for smart card applications," in *Eurocrypt '94*, ser. Lecture Notes in Computer Science, A. D. Santis, Ed., vol. 950. Springer, 1995, pp. 445–449.

[11] D. Naccache, "Method, sender apparatus and receiver apparatus for modulo operation," US patent: US5479511, 1995-12-26, 1993.

[12] D. Naccache, D. M'raihi, W. Wolfowicz, and A. di Porto, "Are crypto-accelerators really inevitable ?" in *Eurocrypt '95*, ser. Lecture Notes in Computer Science, L. C. Guillou and J.-J. Quisquater, Eds., vol. 921. Springer, 1995, pp. 404–409.

[13] A. Arbit, Y. Oren, and A. Wool, "Toward practical public key anti-counterfeiting for low-cost EPC tags," in *International IEEE Conference on RFID*. IEEE, 2011.

[14] Y. Oren and M. Feldhofer, "A low-resource public-key identification scheme for RFID tags and sensor nodes." in *WiSec '09*. ACM, 2009, pp. 59–68.

[15] ——, "WIPR – public-key identification on two grains of sand," in *RFIDSec08 – 4th Workshop on RFID Security*, 2008, http://iss.oy.ne.ro/WIPR.

[16] D. Engels, M.-J. O. Saarinen, P. Schweitzer, and E. M. Smith, "The Hummingbird-2 lightweight authenticated encryption algorithm," in *RFIDSec 2011*, ser. Lecture Notes in Computer Science, A. Juels and C. Paar, Eds., vol. 7055. Springer, 2011, pp. 19–31.

[17] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[18] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Crypto '96*, ser. Lecture Notes in Computer Science, N. Koblitz, Ed., vol. 1109. Springer, 1996, pp. 388–397.

[19] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Crypto '99*, ser. Lecture Notes in Computer Science, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 104–114.

[20] Revere Security, "RVM0CL Version 1.02," Proprietary vendor information, February 2012.

[21] NXP, "LPC1102 32-bit ARM Cortex-M0 microcontroller; 32 kB flash and 8 kB SRAM Rev. 4," Data sheet, June 2011.

[22] ARM, "Cortex-M0+ processor," Website and proprietary information, March 2012.