Balanced Encoding to Mitigate Power Analysis: A Case Study

Cong Chen, Thomas Eisenbarth, Aria Shahverdi and Xin Ye

Worcester Polytechnic Institute, Worcester, MA, USA
{cchen3, teisenbarth, ashahverdi, xye}@wpi.edu

Abstract. Most side channel countermeasures for software implementations of cryptography either rely on masking or randomize the execution order of the cryptographic implementation. This work proposes a countermeasure that has constant leakage in common linear leakage models. Constant leakage is achieved not only for internal state values, but also for their transitions. The proposed countermeasure provides perfect protection in the theoretical leakage model. To study the practical relevance of the proposed countermeasure, it is applied to a software implementation of the block cipher Prince. This case study allows us to give realistic values for resulting implementation overheads as well as for the resulting side channel protection levels that can be achieved in realistic implementation scenarios.

1 Introduction

Embedded implementations of cryptography are a popular target for side channel attacks. With the advent of the Internet of Things, an ever-increasing number of embedded devices enters our lives and homes. These devices handle and exchange possibly sensitive information, raising the need for data security and privacy. High-end security solutions such as the processors found in passports and security smart cards come with an abundance of hardware protection to mitigate all kinds of physical and side channel attacks. However, most embedded devices are consumer-grade products that usually have to rely on unprotected off-the-shelf microprocessors. Only a limited number of methods are available to protect cryptographic software against side channel attacks on such a platform. A popular countermeasure is masking, such as random precharge for registers or full masking schemes [10]. One of the biggest problems for getting a high level of protection of microprocessors is that masking is only effective if the processor has a low signal-to-noise ratio [3, 13]. On modern embedded processors, this is usually not the case, requiring the combination of masking with other countermeasures that decrease the signal-to-noise ratio. Due to the fixed architecture of processors, real hiding countermeasures that achieve leakage reduction are hard to achieve. Proposed countermeasures for embedded software cryptosystems are mostly randomization countermeasures, i.e. leakage is not reduced, but rather randomized in time. Examples include shuffling [17, 19] or random delays [6].

This work explores a true hiding countermeasure in software. The idea is to ensure a constant leakage for all intermediate states. There is some limited prior work proposing constant Hamming weight (HW) encodings of intermediate states. In [9], a secure assembly coding style based on the concept of Dual-Rail Precharge Logic (DPL) was proposed. The authors claim that a constant activity can be achieved using their specific data representation and programming rules. Their work is purely theoretic, no experimental results to support their idea were presented. Furthermore, the computation protocol did not completely prevent Hamming distance (HD) leakage. In [14], the authors present the methods and tools to generate DPL style code automatically. In [8], a similar data representation called Bitwise Balanced enCoding scheme was proposed. This scheme appears to be flawed: the XOR operation will leak information of one of the two inputs, as we explain later. They also just present simulation results that assume an idealized leakage. Hence, that work also lacks any analysis of real-world applicability.

In this work, we present new constant-leakage encodings. As prior works, we require intermediate states to be represented by constant Hamming weight encodings. We go beyond prior studies by showing that requiring constant Hamming distance transitions between states is also feasible. Unlike prior work, we actually implement the counteremeasure, allowing us to realistically judge resulting implementation overheads. More importantly, we evaluate the achieved leakage reduction on a modern 8-bit microcontroller. We show how the constant leakage can be implemented not only for state representations, but also for state transitions, This allows us to apply the encodings to create a protected implementation of the Prince block cipher.

As most countermeasures, this countermeasure cannot provide perfect protection by itself. The leakage of real-world microprocessors deviates from linear and balanced models like the Hamming weight or Hamming distance model. However, forcing the side channel adversary to exploit the non-linear and imbalanced components of the leakage requires more sophisticated attacks and an increasing number of leakage observations. In other words, the countermeasure can effectively decrease the signal-to-noise ratio. The proposed countermeasure is orthogonal to masking or randomization countermeasures. Hence it can easily be combined with those to achieve an even higher overall resistance.

The remainder of the contribution is structured as follows: Related work is discussed in Section 2. The new encoding scheme is introduced in Section 3 and applied to Prince in Section 4. Section 5 explains our leakage evaluation and Section 6 presents implementation results and the outcome of the leakage evaluation.

2 Background

This section introduces the related work on balanced encodings to counteract SCA in both hardware and software implementations, as well as a short introduction to the Prince block cipher to which the countermeasure will be applied.

2.1 Balanced Logic for Hardware Implementation

Dual-Rail Precharge Logic (DPL) aims to achieve constant activity at the gate level. In a DPL style circuit, any gate that generates logic bit A is accompanied with a complementary gate that generates logic bit $\bar{A}$. That is, the logic bit pair $(A, \bar{A})$ is used to represent A. Note that the Hamming weight of the pair is always constant as 1. Besides, in order to obtain constant Hamming distance, the bit pair is precharged to $(0, 0)$ before evaluation. Hence, the gate transitions from $(0, 0)$ to either $(0, 1)$ or $(1, 0)$ leaks data independent power consumption or EM emissions. Based on this idea, many DPL style have been proposed such as WDDL [18], MDPL [12] and DRSL [5]. All these DPL variants aim to protect the hardware crypto systems against SCA.

2.2 Balanced Logic for Software Implementation

Solutions to reproduce DPL in software have been proposed in the past few years. The basic idea of these solutions is the same in that the data in the register or RAM is represented using balanced encoding. Each bit of an intermediate state is converted to two complementary bits. For example, logical bit `1` is encoded as `01` and logical bit `0` is encoded as `10`. `11` and `00` are taken as invalid values.

In [9], Hoogvorst et al. showed a generic assembly coding methodology using DPL style. They redefined the instructions of the standard microprocessor using DPL macros which combines a series of normal instructions to achieve precharge (by moving `0`s to the data register) and evaluation (by concatenating operands and indexing in a lookup table). The activity of precharge phase is constant since overwriting the balanced data register with all $'0'$ causes constant bit flips. During the evaluation phase, the activity of each normal instruction is either constant or irrelevant with the sensitive data. Given the new instructions, the normal assembly code can be transformed to the DPL style code. In [4], Chen et al. also proposed a software programming style to generate a Virtual Secure Circuit (VSC). The basic idea of VSC is to use complementary instructions in a balanced processor to emulate the DPL circuits' behavior.

Han et al. proposed in [8] a balanced encoding scheme for block ciphers. Instead of proposing protection for individual instructions, they propose specific protections for the operations of the cipher, such as KeyAddition and S-box lookups. For example in their KeyAddition layer one balanced encoded key bit (`01` or `10`) is XORed with one plaintext bit. The plaintext bit is encoded by repetition code (`00` for 0 or `11` for 1) so that the result is `01` or `10`, i.e. an internal state bit correctly encoded with a balanced encoding. Obviously, the operation may leak information of the plaintext input, even in the Hamming weight leakage model, but this information is not useful for differential side channel attacks. However, this method can only be applied to the initial KeyAddition where the plaintext is known. For the following rounds, the intermediate data may still leak useful information. Since no alternative XOR is introduced, we do not think this scheme can be applied in an appropriate way to protect cryptographic implementations. Furthermore, the S-box operation also cannot prevent Hamming distance leakage.

2.3 The Prince Block Cipher

The Prince block cipher is a lightweight cipher, featuring a 64-bit block size and a 128-bit key size [1]. Prince has been optimized for low latency and a small hardware footprint. Its round function has several similarities to the AES: it features KeyAddition, S-box, ShiftRows and MixColumns operations. However, these operations are performed on a 4-by-4 array of 4-bit nibbles. This 4-bit oriented design makes Prince—unlike AES—a suitable candidate for a constant Hamming weight encoding on 8-bit microcontrollers. Prince has 12 rounds, and the last six apply the inverse operations of the first six. The 64-bit round key remains constant in all rounds, but is augmented with a 64-bit round constant to ensure variation between rounds. The remaining 64 key bits are used for pre- and post-whitening of the state. A feature of Prince is that encryption and decryption only differ in the round key. A detailed description of an unprotected microcontroller implementation of the Prince can be found in [15].

3 General Balanced Encoding Countermeasure

The non-balanced encoding of the algorithmic inputs and internal states usually causes side channel leakage during the execution of crypto primitives. The leakage can be exploited from classical side channel attacks such as DPA, CPA or MIA. The proposed countermeasure aims at encoding the internal states with longer bit length but resulting in constant Hamming weight of state and constant Hamming distance between two consecutive states. This trade-off sacrifices some memory and efficiency but achieves a balanced representation internal data and therefore mitigates the impact of side channel threats.

Formally, the balanced encoding requires the uniform distribution of Hamming weight for all codewords. Namely, every codeword should have the same Hamming weight, like the idea of constant-weight code or (m of n code). Clearly, the natural binary encoding is not such a candidate (e.g. $HW(0) \neq HW(1)$) since the resulting distribution of Hamming weight is binomial rather than uniform. The idea of balancing encoding can be realized only if using more than necessary bit length. A balanced encoding uses an embedding mapping τ from the natural binary encoded space $\mathcal{C} = \mathbb{F}_2^m$ for all $c \in \mathcal{C}$ into an extension $\mathbf{ext}(\mathcal{C}) = \mathbb{F}_2^n$ with $n > m$. In order to satisfy the constant Hamming weight of the new codeword, a necessary condition is that $C_n^{n/2} \geq 2^m$, where the image $\tau(\mathcal{C})$ sits entirely in the subset $S_{n/2} = \{u \in \mathbb{F}_2^n \mid HW(u) = n/2\}$.

Secondly, the newer encoding should preserve the basic bivariate operations $f(\cdot,\cdot)$ like xor and more complicated univariate operation $g(\cdot)$ such as the non-linear S-box mapping. More precisely, for any $v_1, v_2 \in \mathcal{C}$, it should hold that $\tau(f(v_1, v_2)) = \tilde{f}(\tau(v_1), \tau(v_2))$, where $\tilde{f}$ is the n-bit adjustment of the m-bit operation f. Similarly, for any $v \in \mathcal{C}$, it should hold that $\tau(g(v)) = \tilde{g}(\tau(v))$. Preserving such operations ensures the validity of the algorithmic evolution.

Thirdly, we also want such balanced encoding that achieves constant Hamming distance between any two consecutive states. This may not be easily realized with the choice of the codeword by requiring $HW(\tau(v) \oplus \tilde{g}(\tau(v)))$ being constant for any $v \in \mathcal{C}$. But it can be easily achieved with implementation tricks such as flushing registers before overwriting them with new values. That is, in order to mitigate the leakage generated from overwriting values, say for example, the state representation $\tau(v)$ which is stored in register $R1$ needs to be replaced by the univariate functional output $\tilde{g}(\tau(v))$, the procedure is first to store the output $\tilde{g}(\tau(v))$ at a different pre-cleared register $R2$, then clear register $R1$ and finally copy the register value from $R2$ back to $R1$ and free the temporary register $R2$. This approach sacrifices the efficiency of the code, but prevents Hamming distance leakage from overwriting the state. Another solution is to apply different balanced encodings to the two consecutive states to achieve not only constant Hamming weight but constant Hamming distance as well. More details of this solution will be given in the following section.

4 A Case Study Based on the Prince Cipher

In this section, we use Prince as an example to present the balanced encoding scheme. Prince is a nibble-based block cipher, as detailed in Section 2.3. Since our target platform is an 8-bit processor, a simple balanced encoding can be achieved by simply adding complementary bits, as done for dual-rail logic styles. That way, each state nibble is encoded as a 8-bit balanced encoding by inserting the complementary bits. For any nibble $b_3b_2b_1b_0$ where b_i is one bit data, the complementary nibble is $\bar{b}_3\bar{b}_2\bar{b}_1\bar{b}_0$, where $\bar{b}_i$ is the inverse of b_i. Concatenating these two nibbles forms a balanced encoding $\bar{b}_3\bar{b}_2\bar{b}_1\bar{b}_0b_3b_2b_1b_0$. An alternative is the encoding $\bar{b}_3b_3\bar{b}_2b_2\bar{b}_1b_1\bar{b}_0b_0$. Theoretically, under the Hamming weight leakage assumption, any sequence of those bits can be used as a balanced encoding because the Hamming weight is always 4. In the following we will use two different such encodings, i.e. $enc_I = \bar{b}_3b_3\bar{b}_2b_2\bar{b}_1b_1\bar{b}_0b_0$, which we refer to as *encoding I*, and $enc_{II} = b_0\bar{b}_2b_1b_3\bar{b}_1b_2\bar{b}_0\bar{b}_3$, which we refer to as *encoding II*. Both of the encodings ensure the constant Hamming weight of states. The *encoding II* is used to guarantee the constant Hamming distance between state transitions and the way this specific encoding is determined will be explained in the following section.

KeyAddition with Constant HW/HD In the unprotected Prince implementation, the KeyAddition operation is denoted as $r_3r_2r_1r_0 = b_3b_2b_1b_0 \oplus k_3k_2k_1k_0$ where k is the subkey, b is a state nibble before the KeyAddition and r is the result of KeyAddition. For the protected Prince, we want an XOR-addition where secret inputs and outputs have a balanced encoding. However, for the initial key whitening at the input of the cipher, the plaintext input can be assumed not critical. Hence, only the output r and the key k are mapped to encoding I, i.e. $\bar{r}_3r_3\bar{r}_2r_2\bar{r}_1r_1\bar{r}_0r_0$ and $\bar{k}_3k_3\bar{k}_2k_2\bar{k}_1k_1\bar{k}_0k_0$. As in [8], we can simply XOR-add k in encoding I to b encoded as $b_3b_3b_2b_2b_1b_1b_0b_0$ to realize the partially-protected XOR. This way, the Hamming weight of r is constant as well as the Hamming distance between r and b. The encoding for b does not satisfy the balanced encoding requirement, but has instead double Hamming weight leakage. Therefore, this only works for the initial KeyAddition where the plaintext is known.

After the first KeyAddition, the state becomes sensitive and need the balanced encoding. Hence, for the KeyAddition inside each round, b uses encoding I. Instead, we map k to the encoding $k_3k_3k_2k_2k_1k_1k_0k_0$, resulting in a remaining *constant* leakage for the round keys. Since the leakage is constant, it is not exploitable by CPA or DPA. Note that this leakage can also be avoided by using the XOR addition described in the following MixColumns section. It is more costly than the above described XOR variant, but all inputs and outputs have a balanced encoding and all transitions a constant Hamming distance.

Table Lookup with Constant HW/HD The S-box operation can be described as $s_3s_2s_1s_0 = S(r_3r_2r_1r_0)$ where $S(\cdot)$ denotes the S-box, r denotes an input nibble, and s denotes the output. To protect it, a new lookup table based on the balanced encoding is designed in order to minimize the leakage. The S-box operation is denoted as $\bar{s_3}s_3\bar{s_2}s_2\bar{s_1}s_1\bar{s_0}s_0 = S'(\bar{r_3}r_3\bar{r_2}r_2\bar{r_1}r_1\bar{r_0}r_0)$ where the $S'(\cdot)$ represents the new S-box. Therefore the Hamming weight of S-box output bits is constant. Note that, unlike the regular S-box of size of 1×16, the new S-box is a 16×16 table where the only 16 positions contain the output value and all other positions are unused. The new S-box prevents the Hamming weight leakage but cannot prevent the Hamming distance leakage. One solution is to precharge the target register with zero before writing s into it. An alternative is applying encoding II to s, which is found by exhaustive search in all the possible encodings. For the Prince cipher, the S-box output in encoding II can be denoted as $s_0\bar{s_2}s_1s_3\bar{s_1}s_2\bar{s_0}\bar{s_3}$. In this way, the Hamming weight of S-box output is still constant as 4 and the Hamming distance between input in encoding I and output in encoding II becomes constant as $HD(enc_I(r), enc_{II}(s)) = 4$.

The cost of using two different encodings is an additional reordering layer which coverts encoding II back to encoding I. This is because the following operations such as MixColumns and ShiftRows are based on encoding I. A straightforward idea for reordering is the bit rotation which can be implemented using AND, LSL, LSR and OR instructions. AND instruction is used to pick out each single bit in encoding II by zeroing the other bits. Then we shift it to its position in encoding I. Finally, we combine all bits together to form encoding I. The disadvantage is that it is time consuming and it still causes side channel leakage. Instead, we can implement the reordering layer as a 16x16 lookup table R. The reordering table take the encoding II as input and output encoding $\bar{s_3}s_3\bar{s_2}s_2\bar{s_1}s_1s_0\bar{s_0} = R(enc_{II}(s))$. Note that, the output of R is a variant of encoding I by swapping the two LSBs. This is because $HD(enc_I(s), enc_{II}(s))$ is either 2 or 4 but $HD(\bar{s_3}s_3\bar{s_2}s_2\bar{s_1}s_1s_0\bar{s_0}, enc_{II}(s))$ is constant as 4. Then, the output of R is XORed with $0x03$ which swaps the two LSBs back to encoding I.

MixColumns with Constant HW/HD The MixColumns operation can be implemented as XOR operations between the intermediate data. Unlike the XOR operation in KeyAddition, all the data involved in the MixColumns operation are sensitive and must hence be encoded in balanced encoding scheme to avoid the information leakage. Thus we need to design a new **constant XOR operation** instead of reusing the XOR from the KeyAddition. After the S-box substitution, the data in MixColumns operation are represented in encoding I. Denote the two operands of the constant XOR are as follows: $x : \bar{x_3}x_3\bar{x_2}x_2\bar{x_1}x_1\bar{x_0}x_0$ and $y : \bar{y_3}y_3\bar{y_2}y_2\bar{y_1}y_1\bar{y_0}y_0$. The XOR result is $z : \bar{z}_3z_3\bar{z}_2z_2\bar{z}_1z_1\bar{z}_0z_0$. The constant XOR can be implemented using the following steps:

Step 1: Divide the operand x into two parts and construct two new bytes as $x_L : \bar{x_3}x_3\bar{x_2}x_2\bar{x_3}x_3\bar{x_2}x_2$ and $x_R : \bar{x_1}x_1\bar{x_0}x_0\bar{x_1}x_1\bar{x_0}x_0$. In AVR microcontroller, this step can be easily done by AND, SWAP and OR instructions. For operand y, we construct y_L and y_R in the same way. The following code to the generate x_L can also be applied to the generation of x_R, y_L and y_R.

Input: r1 = x
Output: x_L

```
ldi r16, 0xF0
ldi r17, 0xF0
and r16, r1     ; Cut off the right nibble of x
and r17, r1     ; Cut off the right nibble of x
swap r17        ; Swap the left nibble to the right
or r16, r17     ; Generate x_L
```

Step 2: Do the regular XOR operation between x_L and `0xA5` to generate x'_L : $x_3x_3x_2x_2\bar{x_3}\bar{x_3}\bar{x_2}\bar{x_2} = x_L \oplus (10100101)_b$. Then $z_L = x'_L \oplus y_L = \bar{z}_3z_3\bar{z}_2z_2z_3\bar{z}_3z_2\bar{z}_2$. We also can generate z_R with the similar operations.

Input: r16 = x_L, r18 = y_L
Output: z_L

```
ldi r17, 0xA5
eor r16, r17     ; Convert x_L to x_L'
eor r16, r18     ; Generate z_L
```

Step 3: Combine the most significant nibble of z_L and the least significant nibble of z_R to construct $z : \bar{z}_3z_3\bar{z}_2z_2\bar{z}_1z_1\bar{z}_0z_0$.

Input: r1 = z_L, r2 = z_R
Output: z

```
ldi r16, 0xF0
ldi r17, 0x0F
and r16, r1     ; Cut off the least significant nibble of z_L
and r17, r2     ; Cut off the most significant nibble of z_R
or r16, r17     ; Generate z
```

Note that all above instructions operate on constant Hamming weight representations. Furthermore, there are no transitions that feature a non-constant Hamming distance in any operands. Hence, while costly, this XOR operation is free of Hamming weight or Hamming distance leakages in the operands.

5 Evaluation Methodology

The analyzed countermeasure is secure if each bit of the secret state s leaks in the same way, i.e. linearly and with the same weight. However, practical devices never have such a perfect leakage. To evaluate the leakage properties on the constant weight encoding on a real device, we analyze the leakage behavior of different evaluation approaches. Besides correlation-based DPA, we also perform a mutual information-based evaluation.

5.1 Correlation-Based DPA

Correlation-based DPA was originally proposed by Brier *et al.* [2]. The typical leakage model is the Hamming weight of the S-box output. The studied countermeasure is designed to not feature such a leakage at all. However, since real devices will never feature a perfect Hamming-weight leakage, it is still interesting to analyze whether the remaining leakage of a protected implementation is still exploitable by a CPA. The predicted secret state for our CPA is the Hamming weight of a single S-box output. Another popular analysis is single-bit DPA. As before we apply correlation, but this time using a single bit of the S-box output as leakage model. As the Hamming-weight based CPA, this attack does not work in an idealized environment where each bit leaks in the same way: One of the two bits used to represent the value of a certain bit will always be one, the other zero. However, in practice no two lines leak alike. Hence, bit leakages should be recoverable, but be impeded by the countermeasure.

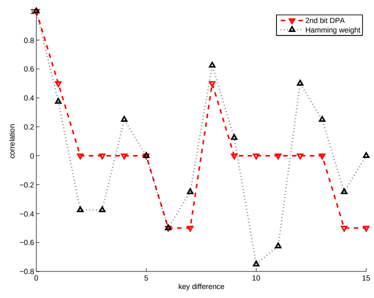


Fig. 1. CPA results on output bit 2 of the unprotected Prince S-box and on the Hamming weight of the unprotected Prince S-box for the correct key (or correctly predicted S-box input, index 0) and incorrect keys (or incorrectly predicted inputs, all other indices). The vertical axis shows the resulting correlation while the horizontal axis indicates the offset from the correct key (or S-box input value). Basically all indices besides 0 that exhibit a significant correlation are considered *Ghost peaks*, showing that Hamming weight based CPA might not be the wisest choice to attack Prince.

Note that both CPA and bit-wise DPA do not behave very well for the Prince block cipher: even in a perfect (Hamming weight) leakage environment, an unprotected implementation features strong "*ghost peaks*", as depicted in Figure 1. These ghost peaks make the distinction of the correct key more difficult. However, since the behavior is predictable, they can also be used to improve the attack, as discussed e.g. in [11].

5.2 Mutual Information based Evaluation

A popular method for evaluating the side channel resistance of an implementation is mutual information. It was proposed as a side channel leakage metric for evaluation in [16] and refined for practical experiments in [7]. The goal is to evaluate the leakage L of a critical intermediate state s. The evaluated intermediate state for the Prince cipher is one nibble. The initial state is a nibble of the plaintext p, which is known. KeyAddition and round constant addition are mere permutations of the state, as are S-box and ShiftRows operations. Since mutual information is computed over all states, the changing labeling does not change the mutual information, i.e. it can be precisely computed through the aforementioned operations even without knowing the key. The MixColumns operation, however, mixes information from different state nibbles, i.e. output nibbles no longer depend on a single input nibble. This means that, during and after the MixColumns operation the meaning of the mutual information that is computed on one nibble finally drops off. However, since the typical target—the S-box output—is fully covered, the leakage typically exploited by any univariate attacks targeting only the first round will be identified by mutual information computed on individual state nibbles. The mutual information $I(S; L)$ between the leakage L and the states S is computed as $I(S; L) = H(S) - H(S|L)$ where $H(S|L)$ is the conditional entropy of S, knowing the leakage L. It is given as

$$H(S|L) = -\sum_{l\in\mathcal{L}} \left(\Pr(l) \sum_{s\in\mathcal{S}} Pr(s|l) \log \Pr(s|l) \right), \tag{1}$$

where l and s are specific observations of the leakage and state, respectively. Given univariate templates $\mathcal{N}(\mu_s, \sigma_s)$ for each state value s and each point of the leakage, we have the probability density for observing a leakage l at that point given as $p(l|s) = \mathcal{N}(\mu_s, \sigma_s)$. Following Bayes' Theorem, we get $p(s|l) = \frac{p(l|s)\Pr(s)}{\Pr(l)}$ and, since all observations and states are equiprobable, we can derive

$$\Pr(s|l) = \frac{p(l|s)}{\sum_{s^*\in\mathcal{S}} p(l|s^*)},$$

as typically done for templates. Plugging this back into Equation (1), we can solve Equation (1) by computing and summing over all $\Pr(s|l^*)$ for each $l^* \in \mathcal{L}_T$, where $\mathcal{L}_T$ is the test (or evaluation) set.

6 Evaluation Results

To verify the balanced encoding scheme, we performed side channel evaluation on three implementations and compared the results between them.

2Prince The first implementation is the unprotected nibble-parallel Prince implementation from [15], in which the 16-nibble states are stored in 8 registers. All round operations process two nibbles in parallel in order to achieve better performance. This implementation feature should result in slightly increased noise if the adversary only predicts a single nibble.

Balanced Prince The second implementation is the protected Prince using encoding I only. In this case, the precharge phase is added to the S-box lookup to achieve not only constant HW but constant HD as well.

Double-Balanced Prince The third one is also the protected Prince but using both encoding I and encoding II. This implementation differs from the second one in that the constant HD is obtained by using encoding II at the S-box output followed by a reordering layer.

We used an 8-bit AVR microcontroller to run the implementations. The performance and memory usage of the implementations are presented below. An automatic power measurement platform was established using a PC, a differential probe and an Tektronix DPO5000 series oscilloscope. A total of 100,000 power traces with random plaintext inputs were obtained for each implementation. Each implementation was analyzed using Hamming weight based CPA as a reference attack. Next, Mutual Information is used as a metric to quantify the leakage and compare the implementations. To make our numbers more reliable, we use 10-fold cross-validation on the computation of the mutual information.

6.1 Implementation Results

First we compare the performance of the three analyzed implementations. Table 1 compares the computation time per encrypted block and resource consumption in terms of code size and RAM usage. The code size increases significantly for the protected implementations, i.e. by a factor of 3. At the same time the performance decreases by a factor of 7. This is because each round operation costs more resources in order to obtain constant activity.

Table 2 shows the contribution of specific operations to the overall resource consumption. In particular, the code size and performance are broken down into the KeyAddition (KA), byte substitution (SB), and the mixing (M) operations of the Prince cipher. For example, the S-box of the protected implementations and the unprotected one are of the same size (256 byte, not included in the table code size calculation), but the unprotected one performs two S-box lookups in parallel. Similarly, either a precharge phase (for the Balanced implementation) or a reordering layer (for the Double-Balanced implementation) had to be added in order to gain constant Hamming distance transitions, also resulting in a significant increase in memory and clock cycles. Additionally, the conversion between normal data and balanced encoded data for the plaintext and ciphertext also adds overhead. The worst overhead is due to the M-Layer, or more precisely the constant leakage XOR, which uses 58 more clock cycles than regular XOR instruction.

Table 1. Performance comparison of the three analyzed implementations.

Implementation	**Encryption Time** in clock cycles	**Code Size** in Bytes	**RAM Usage** in Bytes
2Prince [15]	3253	1574	24
Balanced	28214	3700	472
Double-Balanced	29498	4100	472

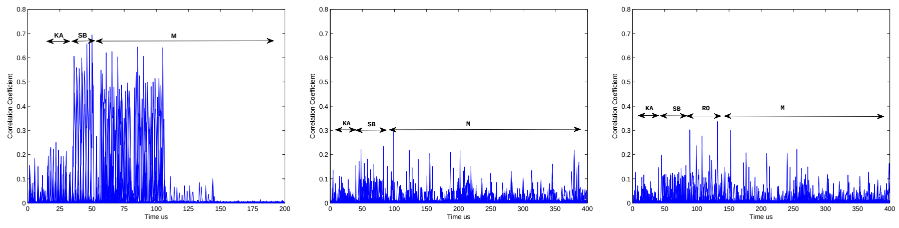
Table 2. Performance and cost comparison for the KeyAddition (KA), byte substitution (SB), and the mixing (M) layers for the three analyzed implementations.

Implementation	**Operation**	**Performance** in clock cycles	**Code Size** in Bytes
2Prince [15]	KA	72	80
	SB	41	36
	M	162	286
Balanced	K	57	68
	SB	90	62
	M	2156	1193
Double-Balanced	KA	57	68
	SB & RO	180	129
	M	2156	1193

6.2 CPA Results

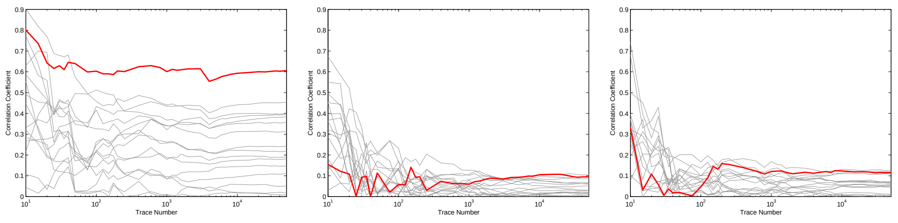
We first performed CPA on all of the three implementations. Each CPA predicts the Hamming weight of the output of a single S-box. To compare the leakage of the implementations—rather than distinguishing the correct key—we use the Hamming weight of the all 16 S-box outputs under a known key as the power model. The results are presented in Figure 2. The correlation between the measurements and power model is greatly reduced in the protected scenarios. For the unprotected implementation, the correlation coefficients range from 0.6 to 0.8 which is only about 0.1 to 0.3 in the protected implementations. Note that a few of the 16 nibbles feature a much stronger leakage than the others in the protected cases (cf. Fig. 2(b) and Fig. 2(c)). This might be an implementation artifact and not due to the countermeasure itself. Similarly, the double-balanced implementation features its strongest leakage in the reordering layer. The results show that the balanced encoding scheme is effective in reducing the Hamming weight leakage. However, due to differences in the leakage of individual bits, the leakage does not completely disappear.

Figure 3 compares the trend of the correlation coefficients of the implementations (vertical axis) over the number of power traces (horizontal axis). We can observe that the correct subkey hypothesis can be easily distinguished from the wrong key guesses with as little as one hundred traces for the unprotected Prince in Figure 3(a). However, for both Balanced Prince and Double-Balanced Prince



(a) 2Prince (b) Balanced (c) Double-Balanced

Fig. 2. Result of CPA of three Prince implementations on the S-box output. The unprotected implementation (a) leaks significantly stronger than the two protected implementations (b) and (c). (KA: KeyAddition; SB: S-box Lookup; RO: reordering M: Mixing Layer)



(a) 2Prince (b) Balanced (c) Double-Balanced

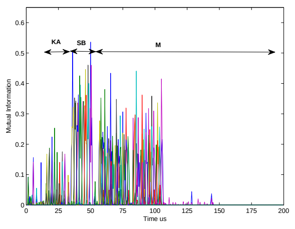
Fig. 3. CPA results for the Hamming weight of the S-box output for the unprotected implementation (a) and the two protected implementations (b) and (c). The vertical axis indicates the absolute value of the correlation coefficient; The horizontal axis indicates the number of traces used. The comparison of the three plots shows the significant improvement resulting from the balanced encodings, if applied correctly. Plot (a) clearly shows the effect of the ghost peaks mentioned in Section 5.

in Figures 3(b) and 3(c), the correlation coefficient is significantly smaller and it is hard to distinguish the correct key hypothesis, even for as many as 50,000 observations. Note that this problem is not obvious in Figure 2, since that figure only contains correlations for the correct subkey hypotheses.

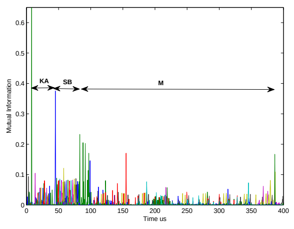
6.3 Mutual Information Based Leakage Analysis

To compare the implementations in a leakage-model independent setting, we apply the mutual information based methodology introduced in Section 5.2 during the first round of the Prince implementation. We apply it in two different ways: First, by using classical univariate templates with an individual mean and variance for each possible nibble state; Next, by using reduced univariate templates with an individual mean for each nibble state, but a common variance for all templates. The latter approach allows to only evaluate first-order leakages.

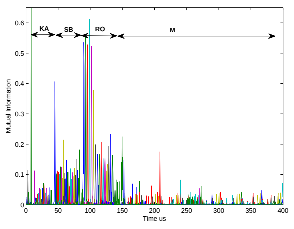
Figure 4 shows the mutual information for all 16 state nibbles for the first round, as derived from full univariate templates. Figure 5 shows the mutual



(a) 2Prince

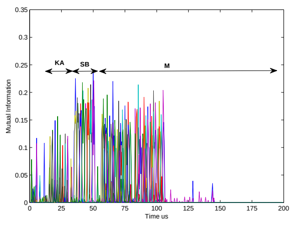


(b) Balanced

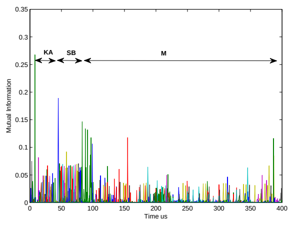


(c) Double-Balanced

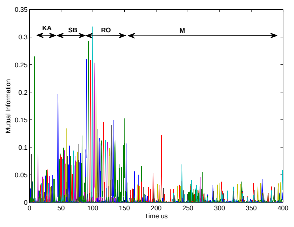
Fig. 4. Mutual information between the state and the leakage for the unprotected (a), Balanced (b), and Double-Balanced (c) implementations during the first round.



(a) 2Prince



(b) Balanced



(c) Double-Balanced

Fig. 5. First order mutual information between the state and the leakage for the unprotected (a), Balanced (b), and Double-Balanced (c) implementations during the first round.

information for all 16 state nibbles only for the first order leakage, as derived from the reduced univariate templates. Both plot families behave very similar, with the first-order MI being slightly lower in all cases. This indicated that the implementations on the AVR feature significant non-linear components in the leakage function. The first-order MI is more appropriate to predict the resistance against first-order attacks such as DPA and CPA. More interestingly, the leakage drops significantly for the protected implementations. In fact, the mutual information goes down by as much as 50%. Especially the leakage of the S-box operation drops even more strongly, from .5 for the unprotected implementation to as low as .1 for the protected ones. That is, there is a single nibble that exhibits a huge leakage for the protected implementations. This is always the first nibble. To remove the leakage, we reordered the nibbles for the computation of the S-box. Surprisingly, whichever nibble is computed first, it exhibits this strong leakage. We claim this to be an implementation artifact. Similarly, there is a leakage right before the KeyAddition starts. Again, we do not have a good explanation for this leakage. However, unlike the S-box leakage, this one is not problematic, as information before the KeyAddition is plaintext, i.e. known to the attacker. As hinted at by the CPA results, both Figures 4(c) and 5(c) show that the Reordering layer still leaks a significant amount of information.

As a result, the Balanced implementation has a weaker leakage than the one of the Double-Balanced implementation. The stronger leakage for the second implementation occurs in the reordering layer. This was not expected, since it is implemented to have a constant Hamming weight and Hamming distance.

In summary, the balanced implementation is a better choice for devices that have a strong Hamming weight leakage and is a valuable new addition to the family of countermeasures in software. The Double-Balanced implementation is slightly less efficient, but suffers from the strong leakage of the reordering layer. A more careful implementation of the reordering layer could reduce the maximum leakage of the Double-Balanced implementation. One should be able to avoid the reordering layer completely by customizing operations in the MixColumns layer, but we did not further explore this route.

7 Conclusion

This work performs the first practical evaluation of the balanced encoding countermeasure in software. While promising in theory, its standalone effectiveness on the modern microcontroller platform used for this study is significant, especially for CPA, but far from perfect. The countermeasure is of high relevance, as it is orthogonal to other software countermeasures such as shuffling and masking, i.e. it can be applied in addition to those. This is of high relevance for platforms that feature high signal-to-noise ratios, such as modern microcontrollers. It is also noteworthy that implementation costs are higher than conjectured, e.g. in [9]. Overall, we believe that this countermeasure technique is useful for lightweight ciphers in cases where additional hiding countermeasure are desirable.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. #1261399 and Grant No. #1314770. We would like to thank the anonymous reviewers for their helpful comments.

References

1. J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin. PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications. In X. Wang and K. Sako, editors, *Advances in Cryptology — ASIACRYPT 2012*, pages 208–225. Springer Berlin Heidelberg, 2012.
2. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 135–152. Springer Berlin / Heidelberg, 2004.
3. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. Wiener, editor, *Advances in Cryptology — CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer Berlin Heidelberg, 1999.

4. Z. Chen, A. Sinha, and P. Schaumont. Implementing virtual secure circuit using a custom-instruction approach. In *Proceedings of the 2010 international conference on Compilers, architectures and synthesis for embedded systems*, pages 57–66, 2010.
5. Z. Chen and Y. Zhou. Dual-rail random switching logic: a countermeasure to reduce side channel leakage. In *Cryptographic Hardware and Embedded Systems-CHES 2006*, pages 242–254. Springer, 2006.
6. J.-S. Coron and I. Kizhvatov. Analysis and improvement of the random delay countermeasure of ches 2009. In S. Mangard and F.-X. Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 95–109. Springer Berlin Heidelberg, 2010.
7. F. Durvaux, F.-X. Standaert, and N. Veyrat-Charvillon. How to certify the leakage of a chip? In *to appear in the proceedings of Eurocrypt 2014*. Springer LNCS, 2014.
8. Y. Han, Y. Zhou, and J. Liu. Securing lightweight block cipher against power analysis attacks. In Y. Zhang, editor, *Future Wireless Networks and Information Systems*, volume 143 of *Lecture Notes in Electrical Engineering*. 2012.
9. P. Hoogvorst, G. Duc, and J.-L. Danger. Software Implementation of Dual-Rail Representation. In *2nd International Workshop on Constructive Side-Channel Analysis ande Secure Design — COSADE 2011*, February 24-25 2014.
10. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smartcards*. Springer-Verlag, 2007.
11. J. Pan, J. G. J. van Woudenberg, J. I. den Hartog, and M. F. Witteman. Improving DPA by Peak Distribution Analysis. In A. Biryukov, G. Gong, and D. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 241–261. Springer Berlin Heidelberg, 2011.
12. T. Popp and S. Mangard. Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints. In J. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 172–186. Springer Berlin Heidelberg, 2005.
13. E. Prouff and M. Rivain. Masking against Side-Channel Attacks: A Formal Security Proof. In T. Johansson and P. Nguyen, editors, *Advances in Cryptology — EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer Berlin Heidelberg, 2013.
14. P. Rauzy, S. Guilley, and Z. Najm. Formally proved security of assembly code against power analysis: A case study on balanced logic. 2013. https://eprint.iacr.org/2013/554.pdf.
15. A. Shahverdi, C. Chen, and T. Eisenbarth. AVRprince - An Efficient Implementation of PRINCE for 8-bit Microprocessors. Technical report, Worcester Polytechnic Institute, 2014. `http://users.wpi.edu/~teisenbarth/pdf/avrPRINCEv01.pdf`.
16. F.-X. Standaert, T. G. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. *Advances in Cryptology — EUROCRYPT 2009*, pages 443–461, 2009.
17. S. Tillich and C. Herbst. Attacking State-of-the-Art Software Countermeasures–A Case Study for AES. In *Proceedings of the 10th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 228–243. Springer, 2008.
18. K. Tiri and I. Verbauwhede. A logic level design methodology for a secure dpa resistant asic or fpga implementation. In *Proceedings of the conference on Design, automation and test in Europe*, page 10246. IEEE Computer Society, 2004.
19. N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In X. Wang and K. Sako, editors, *Advances in Cryptology — ASIACRYPT 2012*, pages 740–757. Springer Berlin Heidelberg, 2012.