A SAT-based Public Key Cryptography Scheme

Sebastian E. Schmittner

Abstract

A homomorphic public key cryptography scheme based on the Boolean Satisfiability Problem (SAT) is proposed. The public key is a SAT formula satisfied by the private key. The probabilistic encryption algorithm generates random Boolean functions, which are implied to be false by the public key. Adding the message bits to them yields the cipher functions. A variant of Blum's Hamilton cycle zero-knowledge proof, adapted to SAT, is used to provide an identification and, via a Fiat-Shamir heuristic, a signature scheme. These are conceptually independent from the encryption scheme.

Index Terms

Public key cryptosystems, Cryptographic protocols, Data Encryption, Message authentication, Digital signatures.

1 Introduction

It is well known that the asymmetric cryptography schemes predominantly used today are vulnerable (at least) to attacks from quantum computers using Shor's algorithm. Assuming P≠NP and that NP-hard problems can not be solved efficiently, not even on a quantum computer, cryptography based on NP-hard problems is dubbed "post-quantum". Daniel Bernstein [1] lists Hash-based, Code-based, Lattice-based, and Multivariate-quadratic-equations cryptography as the existing post-quantum algorithms. The aim of this paper is to introduce a different crypto-system based on the Boolean Satisfiability Problem (SAT). This problem of finding a pre-image of 1 under a Boolean function given in a certain conjunctive form (see Section 2) is well known to be NP-complete. Although a SAT instance can, from a certain angle, be viewed as a multivariate-equation, our encryption scheme is different from the systems mentioned above. In particular, we use a fulfilling assignment as the secret key rather than a trap door.

Using SAT to provide a post-quantum key pair is not a far-fetched idea. The main point of this paper is how to use them to encrypt and decrypt. To this end, we randomly produce Boolean functions which are implied to be false by the public key and hence evaluate to 0 on the private key. These functions $\oplus$ the message bits form the cipher.

The paper is organised as follows: We illustrate the main point of using randomness for encryption in a (somewhat over simplified) picture in Figure 1. The guiding example in Section 1.1 illustrates the main ideas more accurately. In Section 2, we describe and discuss our algorithms in detail. The probabilistic scheme introduced there is vulnerable to an oracle attack, which is discussed in Section 3, as well as some other attacks. In this section we also compute bounds for the parameters of the encryption to resist all mentioned attacks and also explain how the oracle attack can be countered. In Section 4 we introduce an identification and signature scheme, which is independent from the encryption scheme. We conclude in Section 5.

1.1 Example

Before explaining the key generation and encryption algorithm in detail, we start with a simple toy example to guide the readers intuition. A possible[1] private/public key pair for Alice is:

$$\mathit{priv} = (1,1,0,0,1,0,0) \tag{1}$$

$$\mathit{pub} = (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_4 \vee x_5) \wedge (x_1 \vee x_6 \vee x_7) \tag{2}$$

since $\mathit{pub}(\mathit{priv}) = 1$. Bob wants to send a bit $y \in \mathbb{B}$ to Alice. He rewrites

$$\bar{c}_1 = x_1 x_2 x_3 \tag{3}$$

$$\bar{c}_2 = x_1 \oplus x_1 x_4 \oplus x_1 x_5 \oplus x_1 x_4 x_5 \tag{4}$$

$$\bar{c}_3 = 1 \oplus x_1 \oplus x_6 \oplus x_7 \oplus x_1 x_6 \oplus x_1 x_7 \oplus x_6 x_7 \oplus x_1 x_6 x_7 \tag{5}$$

- *Universität zu Köln*
  Institut für Theoretische Physik
  Zülpicher Straße 77; D-50937 Köln
  ses@thp.uni-koeln.de

[1] Of course, the ratio of clauses to variables in realistic keys needs to be much larger (see Appendix E) and also the key length needs to be much longer.

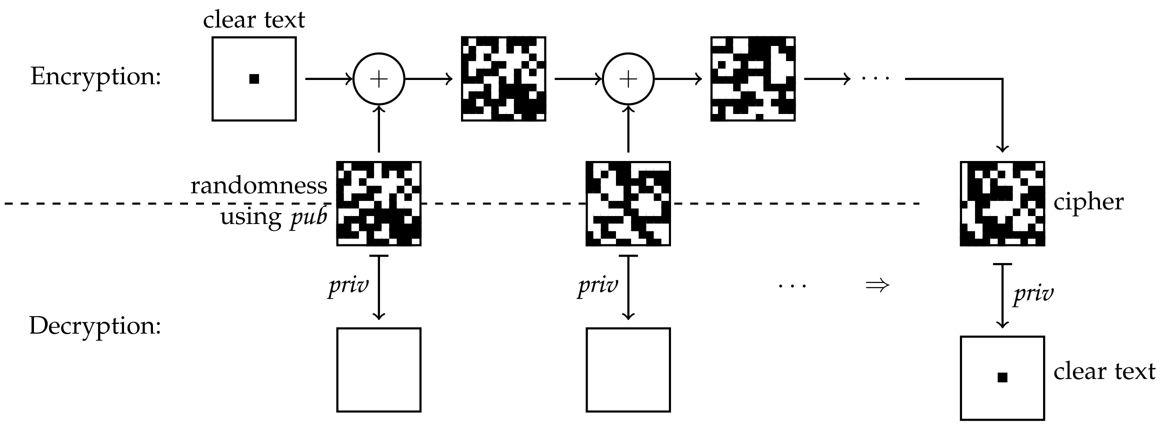

Figure 1. Encryption (top): The text is hidden in random noise patterns generated from the public key *pub*. Subsequent additions hide the structure of the noise patterns to make it harder to subtract the noise.
Decryption (bottom): Via the private key, every noise pattern generated from *pub* is "switched off" and hence the clear text is revealed from the sum.

and randomly generates

$$R_{2,3} = x_4 \oplus x_5 \oplus x_6 \oplus x_4x_5 \oplus x_4x_6x_7 \oplus x_5x_6x_7 \oplus x_4x_5x_6x_7 \tag{6}$$

$$R_{1,3} = 1 \oplus x_2x_3 \oplus x_6x_7 \oplus x_2x_3x_6x_7 \tag{7}$$

$$R_{1,2} = 1 \oplus x_4 \oplus x_5 \oplus x_2x_3 \oplus x_4x_5 \ . \tag{8}$$

The cipher is

$$g = \bar{c}_1 R_{2,3} \oplus \bar{c}_2 R_{1,3} \oplus \bar{c}_3 R_{1,2} \tag{9a}$$

$$= y \oplus 1 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_1x_6 \oplus x_1x_7 \oplus x_2x_3 \oplus x_4x_5 \oplus x_4x_6 \oplus x_4x_7 \oplus x_5x_6 \oplus x_5x_7 \oplus x_6x_7 \oplus x_1x_4x_6 \tag{9b}$$

$$\oplus x_1x_4x_7 \oplus x_1x_5x_6 \oplus x_1x_5x_7 \oplus x_2x_3x_6 \oplus x_2x_3x_7 \oplus x_4x_5x_6 \oplus x_4x_5x_7 \oplus x_4x_6x_7 \oplus x_5x_6x_7 \oplus x_1x_2x_3x_7 \tag{9c}$$

$$\oplus x_1x_4x_5x_6 \oplus x_1x_4x_5x_7 \oplus x_2x_3x_6x_7 \oplus x_4x_5x_6x_7 \tag{9d}$$

Bob sends the cipher g for the clear text y to Alice, who decodes $g(\mathit{priv}) = y$.

Mallet can attack the scheme above by replacing a literal in g by a truth value, say $x_3 \mapsto 1$. He sends the modified cipher to Alice and from Alice's reply, he will notice whether Alice received the correct bit or not. Hence he concludes that $x_3 = 0$ in the private key. Alice might notice the attack by a suitable syntax check on the decoded text (containing more than one bit), but the security of the key pair is lost anyway.

This attack can be countered as follows. Bob seeds his random number generator, which chooses the clauses and generates the random functions R_i, with a hash of the salted clear text. He then sends the salt together with the encrypted message. If the clear text was long enough, it would take Mallet very long to guess the right seed from the salt alone (hence decoding the message). But knowing the salt and the clear text, Alice can easily re-encrypt the message after decryption and check whether the cipher has been altered by Mallet. She rejects any faulty message, regardless of the private key, hence not revealing any information.

2 Algorithms

In this section we specify the algorithms for key generation and encryption. The public key, *pub*, is a "planted" random k-SAT instance for $k \in \mathbb{N}$ with $k > 2$. This is a Boolean function in $n \in \mathbb{N}$ variables, $\mathit{pub} : \mathbb{B}^n \to \mathbb{B}$, which is given as a conjunction of $m \in \mathbb{N}$ many k-clauses, $c_j : \mathbb{B}^k \subset \mathbb{B}^n \to \mathbb{B}$, together with a truth assignment, the private key $\mathit{priv} \in \mathit{pub}^{-1}(1)$. In summary

$$\mathit{pub} : x \mapsto \bigwedge_{j=1}^{m} c_j(x) \ , \qquad\qquad c_j : x \mapsto \bigvee_{i=1}^{k} (x_{I(i,j)} \oplus s(i,j)) \ , \tag{10}$$

where the signs $s(i,j) \in \mathbb{B}$ determine wether the literal x_I is negated. To discuss the complexity of the algorithms, we use the notation

$$O(f) := \left\{ g : \mathbb{N} \to \mathbb{R} \mid \limsup_{n\to\infty} \frac{|g(n)|}{f(n)} < \infty \right\} \ , \qquad\qquad o(f) := \left\{ g : \mathbb{N} \to \mathbb{R} \mid \limsup_{n\to\infty} \frac{|g(n)|}{f(n)} = 0 \right\} \ , \tag{11}$$

and $\Theta(f) := O(f) \setminus o(f)$ for $f : \mathbb{N} \to \mathbb{R}^+$.

2.1 Key Generation

A key pair is generated as follows:

1) Choose the private key *priv* $\in \mathbb{B}^n$ at random (equidistributed).
2) Generate the clause c_j by randomly choosing k distinct integers $I_{j,i} \in \{1, \ldots, n\}$ and "signs" $s_i \in \mathbb{B}$.
3) If $c_j(\textit{priv})$ accept the clause, otherwise reject it.[2]
4) Repeat until m clauses have been generated.

In other words, the data to be generated and stored for the each key pair is

$$I : \{1, \ldots, m\} \times \{1, \ldots, k\} \to \{1, \ldots, n\} \qquad \text{and} \qquad s : \{1, \ldots, m\} \times \{1, \ldots, k\} \to \mathbb{B} \;, \tag{12}$$

hence the length of the public key is in $\Theta(mk(\log(n) + 1))$. The run time is proportional to the public key length[3].

It is very important to generate hard instances as public keys. Therefore we must ensure that *priv* is (almost surely) the unique solution, i.e. m needs to be larger than the critical value, $m > m_c = \alpha_c(k)n$ for some $\alpha_c(k) \in \Theta(1)$, see [4]. Choosing m rather close to m_c yields the most difficult instance. This is well known in similar setups, see Appendix E for some of our own benchmarks. Choosing $m \propto n$ for fixed k leads to a public key length of $\Theta(n \log(n))$. A different scaling of m with n is not recommended, since the resulting SAT instances then become much easier to solve.

2.2 Encryption and Decryption

We fix parameters $\alpha, \beta \in \mathbb{N}$ with $2 \leq \beta \ll \alpha$ to tune the run time and security of the encryption.

1) Choose α many tuples of β many distinct clauses at random, $J : \{1, \ldots, \alpha\} \times \{1, \ldots, \beta\} \to \{1, \ldots, m\}$ such that $J(i, a) \neq J(i, b)$ for all $a \neq b$.
   a) To counter attacks discussed in Section 3.1.2, it is preferable if the clauses within one tuple share variables. This is particularly important if one of the clauses does not contain negations, i.e. $s(i, 1) = \ldots = s(i, k) = 0$.
   b) We have to ensure that each tuple shares at least one clause with another tuple to counter the attack discussed in Section 3.2.
   c) Each clause of *pub* is to appear in some tuple as discussed in Section 3.3.
2) The cipher encoding $y \in \mathbb{B}$ is the algebraic normal form (ANF) of
$$g = y \oplus \bigoplus_{i=1}^{\alpha} \bigoplus_{a=1}^{\beta} \bar{c}_{J(i,a)} \wedge R_{i,a} \;. \tag{13}$$
   Here $R_{i,a}$ is a random function. It depends on the same set of variables as the clauses $\{c_{J(i,b)} | b \neq a\}$ and is generated in ANF where each possible term occurs with probability $1/2$.
3) The decryption "algorithm" is $g(\textit{priv}) = y$.

We can ensure (1b) and (1c) by choosing J as follows: First choose a permutation $\sigma \in S_m$ at random. Then set $J(i, a) = \sigma(i + a - 1)$, where the indices are understood modulo m. In this scheme $\alpha = m$. To also ensure (1a) one should enhance the probability of neighbouring clauses, $c_{\sigma(j)}$ and $c_{\sigma(j)+1}$, to have variables in common when generating σ.

To turn the negated clauses in (13) into ANF, one can use various identities, such as $x \vee y = x \oplus y \oplus xy$, but $\overline{x \vee y} = \bar{x}\bar{y}$ is the most efficient one here. It leads to

$$1 \oplus (x_1 \oplus s_1) \vee \ldots \vee (x_k \oplus s_k) = (x_1 \oplus s_1 \oplus 1) \wedge \ldots \wedge (x_k \oplus s_k \oplus 1) \;. \tag{14}$$

The resulting ANF (obtained by distributing $\wedge$ over $\oplus$) is of length $\leq 2^k$. The run time to compute the cipher in (13) as well as the length of the resulting cipher are in $O(\alpha 2^{\beta k})$. Consequently, we need to choose $\beta \in O(\log(m))$ in order for the run time to be polynomial. The length can be expected to be shorter than the run time by a factor > 2 due to the cancellation of terms in the sum and due to $R_{i,a}$ only containing $2^{(\beta-1)k-1}$ terms on average. If we choose $\beta = \log_2(m)/k$ and $\alpha \propto m$, we have $\alpha 2^{\beta k} \propto m^2$. For $\beta \in \Theta(1)$, run time and cipher length formally only scale with m, but for $m \approx 2^{10}$ and $\beta = k = 3$ the pre-factor is still comparable to m. In short, encryption with this setup has complexity $O(m^2)$.

3 Known Attacks and Improved Schemes

Denote by $G_b = G_b(\textit{pub})$ the set of all possible ciphers encoding $b \in \mathbb{B}$ and by $G = G_1 \cup G_0$ the set of all ciphers. Since the problem of deriving the private from the public key is known to be hard, we will focus on attacks on the cipher. We will refer to the problem of deciding for $g \in G$ whether $g \in G_0$ or $g \in G_1$ as the decoding problem. We do not have a proof that this is a hard problem, although some related problems are (see Appendix A). In this section we focus on some particular attacks, i.e. algorithms that solve the decoding problem and show that all of them have exponential run time, if the parameters of the encryption algorithm are chosen suitably.

[2] If the algorithm is to have finite worst case run time, one should modify the clause instead of generating a new one at random. Notice, however, that the number of signs to flip has to be chosen carefully if the resulting distribution of public keys is to be unchanged. See e.g. [5].

[3] If clauses are rejected this is tha case with probability exponentially close to 1.

3.1 Simple Attacks

3.1.1 Enumeration of Ciphers Attack

Since the set of all possible ciphers for a given public key is finite, a trivial brute force attack is to just enumerate it. In order to prevent this attack, the encryption parameters α and β need to be large enough. More precisely, the number of possible ciphers is roughly

$$\binom{m}{\beta}^{\alpha} 2^{\alpha\beta((\beta-1)k)} \ . \tag{15}$$

We can ensure this number to be (super) exponentially large by choosing

$$n \in O(\alpha\beta) \ . \tag{16}$$

Further we should apply some fixed invertible linear transformation to the message bit vector before encoding. This way, decoding a few of the cipher bits does not reveal any clear text bits.

3.1.2 Constant Term Probability Attack

Another rather trivial attack is to determine $g(0,\ldots,0)$ for a cipher $g \in G$. This is just the presence or absence of the constant 1 in the ANF, which is where the message bit enters. Hence this property must not distinguish G_0 from G_1.

The ANF of a random clause of length k contains a constant 1 with probability $1-2^{-k}$, hence the probability for each summand in (13) to contain the constant 1 is 2^{-k-1}. This is rather small, but a variant of the central limit theorem is in this case on our side, see Appendix B. As a consequence, the probability for $g(0,\ldots,0)=1$ tends to $1/2$ exponentially quickly with $\alpha\beta$ at the scale of

$$\alpha\beta \gg 2^k \ . \tag{17}$$

Considering the enumeration attack from Section 3.1.1, the bound established in (16) is much stronger than (17). In other words, the encryption resists the constant term attack if parameters are chosen such that it resists the enumeration attack.

A more refined version of this attack focuses on those clauses which can contribute a summand 1, namely those c_i which do not contain any negation. For those,

$$\bar{c}_i = (x_{i_1} \oplus 1)\ldots(x_{i_k} \oplus 1) = 1 \oplus x_{i_1} \oplus \ldots \oplus x_{i_1}x_{i_2}\ldots x_{i_k} \tag{18}$$

contributes a constant to the cipher sum iff the corresponding R_i also contains the constant 1. An attacker can try to judge wether the latter is the case by looking for the order k term, $x_{i_1}x_{i_2}\ldots x_{i_k}$, in the cipher sum. The same term could be caused by another clause depending on the same variables, but such a clause is part of *pub* with vanishing probability for large $m \propto n$. Hence we need to ensure that c_i shares at least one variable with another clause in the same tuple or that another clause in the same tuple contains no negated literals. Both of these can lead to the appearance of the same order k term, see the example in Section 1.1.

3.1.3 Cipher Value Probability Attack

An attacker can try to estimate $|\{g^{-1}(1)\}|$ for any $g \in G$ by evaluating g on a random set of trial inputs. Hence this value must not distinguish G_1 from G_0. This attack is structurally very similar to the constant term attack (Section 3.1.2). For each clause c of length k, the number of inputs on which c is 1 is $|\{c^{-1}(1)\}| = 2^k - 1$. The probability of *all* summands in the cipher g in Equation (13) evaluating to 1 on a random input is the same as for none of the summands to contain a constant 1, if $R = 1$ with probability $1/2$. Even if the Rs are chosen such that they evaluates to 1 only on $\propto 2^{-\beta k}$ many inputs, choosing

$$\alpha \gg k \tag{19}$$

still ensures that the probability for $g(1) = 1$ is exponentially close to $1/2$ according to Appendix B. Since $\beta \in O(\log(m))$ for run time reasons, condition (19) is again ensured by (16).

3.2 Decoding Attack

More sophisticated attacks exploit the structure of the encoding algorithm. Any function in $g \in G$ is necessarily of the form

$$g = y \oplus \bigoplus_{i=1}^{m} \bar{c}_i R_i$$

for some functions R_i (see Appendix C). A crucial point of our whole scheme is that polynomial long division does *not* work over finite fields such as $\mathbb{B}$, because the degree is not well behaved under addition and multiplication in the polynomial ring. This can be seen in the Example 1.1. A simpler example is $x^{n+1} = x$ and hence e.g. $x^2y \oplus y^2x^5 = 0$. Such "collisions" are sufficiently likely to secure our ciphers, if the random functions R_I are chosen in a good way.

Attacks on the structure of the encoding algorithm start with the following observation. The set of all clauses which depend only on literals in a small set $\{x_{i_1}, \ldots, x_{i_M}\}$ has an expected size of

$$\binom{M}{k} \frac{m}{n^k} , \tag{20}$$

which is exponentially small for $M \in o(m)$. In other words, a small set of clauses C is most likely uniquely determined by $D(C) := \bigcup_{c_i \in C} D(c_i)$ where $D(c(x_1, \ldots, x_k)) = \{x_1, \ldots, x_k\}$ denotes the variables on which c depends. This means that most likely all tuples $J(i, 1), \ldots, J(i, \beta)$ used to generate our cipher g in Section 2.2 can be identified from the ANF of g. For a given tuple, this leaves $2^{\beta(\beta-1)k}$ choices for the random functions associated with it. If we choose $\beta = \log_2(m)/k$ (see Section 2.2) then $2^{\beta(\beta-1)k} \approx m^{\log(m)}$. This is super polynomial in m, but, for reasonable values of m, an attacker can still produce all possible summands associated with this tuple and try to add them to g. He can check whether all terms involving variables from this tuple cancel. However, the same variables will occur also in other tuples. In fact, we should ensure that also some of the *clauses* from this tuple appear in other tuples. This will leave the attacker with a few possibilities for the random function, which can only be decided once the touples sharing clauses are decided as well. But since all tuples are connected (indirectly) by sharing clauses, the run time of this attack is exponential in α which we chose to be of the order of m.

3.3 Reduced SAT Problem Attack

If an attacker can learn that only a certain set of clauses was used to produce a cipher, he can try to solve the SAT problem given by the conjunction of only those clauses that were used. If the ratio of used clauses to variables appearing in those clauses is small enough, the SAT problem is easily solvable and any solution can be used to decode the cipher.

To resist this attack, care is to be taken in Step (1) of the encryption (Section 2.2). The set of all clauses used, $C = \bigcup_{i=1}^{\alpha} \bigcup_{a=1}^{\beta} \{c_{J(i,a)}\}$ should not depend on more than $|C|n/m$ many different variables. Furthermore, reducing the number of clauses used, even at constant clauses to variables ratio, effectively reduces the key length. Overall, we should ensure that (almost) all clauses are used to generate the cipher.

3.4 Oracle Attack

The communication according to the algorithms from Section 2.2 is vulnerable to the following attack: If the recipient is expected to send a reply to an encrypted message, an attacker can fake a cipher by replacing a literal with a guessed truth value. If he can learn from the reply whether the cipher was correctly decoded, he gets a strong hint, or even evidence, for the assignment of this literal in the private key. Repeating the attack in a suitable scheme will reveal the full private key. This is a severe attack which limits the scope of application of the random encryption algorithm to such cases where replies are only send to authenticated communication partners. Post-quantum authentication could be provided by e.g. a hash-based scheme, see [1, Hash-based Digital Signature Schemes]), or by our identification scheme introduced in Section 4. One could also use only one-time key pairs for encryption, again with the problem of authenticating the new keys.

Instead, we have developed two improved version of our scheme. The one which we consider superior resist the oracle attack completely and is described in Section 3.4.1. In special circumstances, also the version described in Appendix D might be useful. The latter makes the oracle attack substantially more difficult and preserves the probabilistic nature of the encryption.

3.4.1 Proof of Honest Encryption

The version of the encryption/decryption scheme described here prevents the oracle attack completely without increasing the complexity of the encryption algorithm. Key generation is unchanged, but decryption becomes as complex as encryption and the feature of stochastic ciphers is lost. More precisely, using the encryption discussed in Section 2.2, the cipher is not a function of the public key and clear text. In particular, even if Bob encodes the same clear text twice, he will get different ciphers. This makes repetition-attacks impossible and the algorithm is very resistant against rainbow table type attacks. It is, of course, only pseudo random. In other words, the cipher *is* a function of public key, clear text *and the seed of the pseudo random number generator* (PRNG), implicitly used to make the random choices. This can be used to verify that the cipher was not tempered with in order to oracle the private key, at the cost of loosing the rainbow-resistance. The latter can then be restored in the usual way by salting.

Concretely, key generation as described in Section 2.1 stays untouched and also encryption is done as explained in Section 2.2, but the sender starts by seeding the PRNG with a specific seed, computed from clear text and a salt. The salt is then to be part of the cipher. After decryption of the cipher (applying it to the private key), the recipient computes the seed from the clear text and the salt. He then checks that the received cipher text matches that one that is computed from the public key, the clear text, and the seed according to the fixed encryption algorithm. This way, oracle attacks can be detected without revealing any information.

The cost to pay is that the implementation of the PRNG and the encryption algorithm on the sender and receiver side have to match and that the receiver has to re-do the most time consuming part of the whole scheme, the encryption. Further more, the clear text needs to be long enough (of order n) in order to prevent a brute force attack on the seed. Notice that the seed, as computed from the salt *and clear text*, is to be considered as a key for decoding the particular message. If an attacker can find the seed he can decode the cipher without the private key.

4 Identification and Signatures

In this sections we introduce a zero-knowledge proof for our key pairs. This leads to an identification and signature scheme, which is independent from the encryption scheme discussed above, apart from sharing the same type of public/private key pairs. In fact, the scheme proposed here is a variant of the one by Blum based on the Hamitlon cycle problem [2]. The latter is NP-complete, hence polynomial-time equivalent to SAT, but one can apply the ideas more directly.

4.1 Zero-Knowledge Proof/Identification

Let pub again be a k-SAT instance in n boolean variables and $\mathit{priv} \in \mathit{pub}^{-1}(1)$ a solution only known to Alice. The zero-knowledge proof consists of K rounds, each of which proceeds as follows:

1) Alice chooses a random invertible function $f : \mathbb{B}^n \to \mathbb{B}^n$ and commits to $f^*(\mathit{pub}) = \mathit{pub} \circ f$ and to $f^{-1}(\mathit{priv})$. More precisely, the commitment to $f^*(\mathit{pub})$ is to be in the form of committing to each literal in the pull back of the form, i.e. to each $f^*(x_I(1,1)), f^*(x_I(1,2)), \dots, f^*(x_I(m,k))$.
2) Bob chooses (randomly) whether Alice should reveal either
   a) all of $f^*(\mathit{pub})$ or
   b) $f^{-1}(\mathit{priv})$ together with $\left\{f^*(x_{I(i,a_i)}) | i \in \{1,\dots,m\}\right\}$, where Alice randomly chooses the $a_i \in \{1,\dots,k\}$ such that $f^*(x_{I(i,a_i)})\left(f^{-1}(\mathit{priv})\right) = 1$ for all i.

In this scheme Bob either verifies that Alice did not cheat when generating $f^*(\mathit{pub})$ or that she indeed knows a solution. Fake proofs will hence be discovered with probability $1/2$ each round and the probability to get through all rounds is 2^{-K}.

The run time of the above scheme depends on what kind of functions we allow for f. If we only use permutations[4], the run time for generating the permutation, its inverse and reshuffling priv is $O(n)$ and $O(m)$ for re-labelling pub. However, in this case the number of 1s and 0s in the public key is leaked by revealing $f^{-1}(\mathit{priv})$. To avoid this, we should at least add a random affine shift $s \in \mathbb{B}^n$ to the permutation, i.e. $f(x) = \sigma(x) + s$.

4.2 Signatures

By a Fiat-Shamir heuristic we can convert the above interactive identification into a signature scheme. To this end, let $h : \mathbb{B}^* \to \mathbb{B}^K$ be a cryptographic hash function. To sign a document:

1) Alice generates K random functions, $f_1, \dots, f_k$, as in Section 4.1 and commits to all of the $f_i^{-1}(\mathit{priv})$ and $f^*(\mathit{pub})$.
2) She then computes a hash of the document to be signed, concatenated with (a suitable encoding of) her commitments. Denote the latter by C.
3) She reveals the information as in Step 2 of Section 4.1, choosing 2a or 2b according to the obtained hash. Denote this revealed information by R.

The signature consists of C and R. It is verified by Bob by checking that the correct information was revealed by recomputing the hash. He also checks that the revealed information is valid as in Section 4.1, of course.

To fake this signature, Alice could fake the zero-knowledge proofs and generate signatures until, by chance, a valid one is generated. However, the probability for this to happen is as small as cheating in the identification scheme, i.e. exponentially small in the number of rounds.

5 Conclusion

We have introduced a public key crypto-scheme based on the Boolean Satisfiability Problem. The motivation for developing such an algorithm is to fill the arsenal of post-quantum cryptography with some fresh ammunition. The simplicity of the scheme developed in this paper might be an advantage over the well known post-quantum schemes. More conceptually, we are not aware of any scheme which uses random ciphers in a similar way (compare to Figure 1). Furthermore, there is quite some freedom in the details of the encryption algorithm, in particular in choosing probability distributions for the random functions R_I. This makes it possible to adapt the algorithm, if more sophisticated attacks are discovered in the future. The encryption presented in Section 2.2 has undergone some evolution to resist all attacks which came to our mind. The version using a proof honest encryption has no vulnerabilities currently known to us, but of course much more crypto-analysis is needed and the reader is invited to devise stronger attacks to challenge and improve the algorithm. In particular, we have not proven that it is (NP-)hard to decipher a message without knowledge of the private key. It is the problem of deducing the private from the public key that is NP-hard, i.e. "post-quantum", by construction. Another notable feature of our encryption is that it is (perfectly) homomorpic, i.e. applying any boolean function to the cipher bit vector and then decoding yields the same result as decoding and then applying the function.

The oracle attack mentioned in Section 3.4 is a severe generic attack on any cipher consisting of Boolean functions. Enforcing honest encryption as explained in Section 3.4.1 is a generic counter. The multi-key version of our scheme described in Appendix D is another (generic) work around. However, this one is not completely resistant against oracle

[4] Here the permutation acts on $\mathbb{B}^n$ by permuting the coefficients with respect to the fixed base and correspondingly on formulas by permuting the indices of the variables.

attacks. Although it only reveals much less information, the key pairs still need to be changed regularly in the multi-key scenario. As opposed to the simple version, here this could here be feasible. In some special situations, the multi-key version might still be preferable, since encryption is considerably more complex than key generation and decryption which in particular yields some protection against DOS attacks for multi-key schemes.

The length of the public key and the run time of the key generation algorithm scale as $O(n \log n)$ with the length of the private key n. The length of the cipher and the run time of the encryption algorithm per bit of clear text scale as $O(m^{1+\epsilon})$ with $\epsilon \geq 0$ and we have identified $2\alpha_c(k)n > m > \alpha_c(k)n$ as the relevant parameter range for hard planted SAT instances (see Appendix E). The crucial question is how big n should be in practice today. At the SAT Competition 2014[5], random 3-SAT instances of size $n \approx 10^4$ have been solved for $m \approx m_c$ in less than one hour.[6] These instances were selected in order to be solvable within that time, but still they are randomly generated with non-negligible probability. For $m < m_c$ even instances with $n \approx 10^6$ have been solved, but these instances have most likely very many solutions and are hence much easier than our public keys. Our own benchmarks (Appendix E) show that the MiniSat solver [3] can not break keys of length $n > 2^{10} \approx 10^3$ on a modern PC using one thread. Taking large scale parallelisation into account and in view of the SAT Competition results, one should choose $n > 2^{23} \approx 10^7$.

Key generation is very fast and choosing $n > 10^7$ is not a problem here. However, encryption with our not very much optimised proof of concept implementation [6] already takes time of the order of few seconds per bit of clear text to encode with $\alpha = m = 5n = 5 \times 2^{10}$ and $\beta = 3$ and still tenth of seconds per bit for $\beta = 2$ and the other parameters as before. Although there is certainly room for improving the implementation, the constraint $\beta \geq 2$ means that the run time of the encryption algorithm is in $O(m^{1+\epsilon})$ with $\epsilon \geq 3/5$ for $m \approx 2^{10}$. These run times are not yet very well suited for practical applications. However, the situation could be improved by advancing away from bit-wise encryption. Encoding longer messages at once is an interesting topic for future research. If one can counter the attack from Section 3.3 by other means than using all clauses, smaller α would also lead to a substantial speed up. Anyway, before more research builds on the ideas presented here, we would like our algorithm to be challenged by more advanced attacks in order to improve it. Furthermore, the problems which we have proven to be NP-hard (see Appendix A) are still not very close to the decoding problem. To build faith in this encryption really being post-quantum secure, more research in this direction is in order.

The signature scheme presented in Section 4.2 is independent of our encryption scheme. It is rather close to Blum's well known scheme [2], but adapted to our SAT scenario and post-quantum secure by construction.

Acknowledgement

We acknowledge helpful discussions with Rainer Schrader and Ewald Speckenmeyer as well as their research groups in Cologne. In particular, we would like to thank Martin Olschewski, who pointed out the oracle attack.

We are very thankful to Paulo Barreto for friendly advice and his interest in our first draft of this paper. In particular his suggestion to introduce signatures via a zero knowledge proof brought this idea home to us.

The author was supported by Martin Zirnbauer, by a grant from Deutsche Telekom Stiftung, and another grant from the Bonn-Cologne Graduate School of Physics and Astronomy, funded by the DFG.

References

[1] D. J. Bernstein, J. Buchmann, and E. Dahmen, editors. *Post-Quantum Cryptography*. Springer-Verlag Berlin Heidelberg, 2009.

[2] M. Blum. How to prove a theorem so no one else can claim it. *ICM Proceedings*, 2:1444–1451, 1986.

[3] N. Een and N. Sorensson. Minisat. http://www.minisat.se/.

[4] S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random boolean expressions. *Science (New York, N.Y.)*, 264(5163):1297–1301, 1994.

[5] F. Krzakala, M. Mezard, and L. Zdeborova. Reweighted belief propagation and quiet planting for random k-SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 8:149–171, 2014.

[6] S. E. Schmittner. A prove of concept implementation of a SAT-based encryption scheme. https://github.com/Echsecutor/kryptoSAT, 2015.

[5] http://www.satcompetition.org/2014/

[6] E.g. http://satcompetition.org/edacc/sc14/experiment/24/result/?id=23531

Appendix A
Some Hard Problems

In this section, we establish that some problems related to the decoding problem (see Section 3) are hard. To this end, fix a public key *pub* and assume that $\mathit{pub}^{-1}(1) = \{\mathit{priv}\}$. The partitioning of ciphers $G = G_0 \cup G_1$ is characterised by

$$G_1 = \{g \in G \mid \mathit{pub} \Rightarrow g\} \tag{21}$$

$$G_0 = \{g \in G \mid \mathit{pub} \Rightarrow \bar{g}\} \; . \tag{22}$$

Deriving the private key is harder than decoding, since $\forall g \in G : g \in G_{g(\mathit{priv})}$. Assuming that all $g \in G$ are given in such a form that evaluating $g(\mathit{priv})$ is possible in polynomial time, the decoding problem is in NP.

If G would contain the elementary functions $X := \{x \mapsto x_i\}$ then the decoding problem would be (polynomial time) equivalent to deriving the private key, hence solving the SAT problem. A set of functions Y of polynomials size will be called "hard to decode", if solving the decoding problem for each $f \in Y$ determines (by a polynomial time algorithm) the solution of the SAT problem. The following sets of functions are hard to decode:

1) The elementary functions X
2) Any set of functions containing a subset that is hard to decode
3) $\{f \oplus s_f | f \in Y\}$ where $s_f \in \mathbb{B}$ are known and Y is hard to decode
4) $\{f \oplus g | f \in Y\}$ where Y is hard to decode and g is any Boolean function
5) $\{f \wedge g | f \in Y\} \cup \{f \wedge \bar{g} | f \in Y\}$ with Y, g as above
6) $\{x_i \wedge f | f \in Y, x_i \in X\}$ with X, Y as above
7) $\{x_i \vee f | f \in Y, x_i \in X\}$ with X, Y as above

(3) is hard to decode since $f \oplus s \in G_b \leftrightarrow f \in G_{b \oplus s}$. (4) is hard to decode since we can decide for each $f \oplus g$ whether or not it is implied by *pub*, then assume that $g(\mathit{priv}) = 0$ and use that Y is hard to decode to produce a trial solution priv'. If $\mathit{pub}(\mathit{priv}') \neq 1$ then $g(\mathit{priv}) = 1$ and we can use (3). (5) is hard to decide since one of the two sets evaluates to $\{0\}$ on *priv*. Since there are only two possibilities for $g(\mathit{priv})$ one can check both in polynomial time, similar to (4). In (6) (and similarly in (7)) we can decide $\{x_1 \wedge f\}$. If this turns out to be a subset of G_0 then we likely have $x_1(\mathit{priv}) = 0$ and we proceed deciding $\{x_2 \wedge f\}$. After polynomial time we either arrive at some $x_i(\mathit{priv}) = 1$ and can hence derive *priv* or conclude that $\mathit{priv} = (0, \ldots, 0)$.

The above construction shows that some generic problems similar to the decoding problem are hard to decide. Notice, however, that the sender can not (on purpose) encode a message of polynomial length into a hard to encode set of functions in polynomial time, or else he would solve the SAT problem.

Appendix B
Boolean Central Limit Theorem

Consider a set of i.i.d. Boolean variables x_i, with $p_1 := \mathrm{prob}(x_i = 1)$. Then

$$p_0^{\oplus} := \mathrm{prob}\left(\bigoplus_{i=1}^{M} x_i = 0\right) = \sum_{j=0}^{\lfloor M/2 \rfloor} \binom{M}{2j} p_1^{2j} (1 - p_1)^{M-2j} \tag{23}$$

$$p_1^{\oplus} := \mathrm{prob}\left(\bigoplus_{i=1}^{M} x_i = 1\right) = \sum_{j=0}^{\lceil M/2 \rceil - 1} \binom{M}{2j+1} p_1^{2j+1} (1 - p_1)^{M-2j-1} \tag{24}$$

and hence $|p_0^{\oplus} - p_1^{\oplus}| = |1 - 2p_1|^M$ converges to 0 for any $0 < p_1 < 1$ and $M \to \infty$ at exponential speed. In other words, for large enough M we have $p_1^{\oplus} \approx 1/2$ irrespective of p_1. Large enough here means $M \gg -1/\log|1 - 2p_1|$, which means $M \gg p_1^{-1}/2$ for small $p_1 \ll 1/2$.

Appendix C
Structure of Ciphers

In this section we discuss which functions g can, in principle, be constructed from the public key *pub* without knowledge of the private key, such that $\mathit{pub} \Rightarrow g$, i.e. $g \in G_1$. We will therefore assume that only the clauses c_i of $\mathit{pub} = \bigwedge_{i=1}^{m} c_i$ can be used as the elementary building blocks for which $\mathit{pub} \Rightarrow c_i$ is known.

Any encryption algorithm of the type discussed in this paper will lead to sets of ciphers G and G_b which are contained in the sets $\tilde{G}$ and $\tilde{G}_b$, respectively. The latter are constructed as follows

1) $1 \in \tilde{G}_1$ and $c_i \in \tilde{G}_1$ for $i \in \{1, \ldots, m\}$.
2) $f \in \tilde{G}_a, g \in \tilde{G}_b \Rightarrow f \oplus g \in \tilde{G}_{a \oplus b}$, $f \wedge g \in \tilde{G}_{a \wedge b}$ and $f \vee g \in \tilde{G}_{a \vee b}$. In particular, $\bar{g} = 1 \oplus g \in \tilde{G}_{\bar{b}}$.
3) $f \in \tilde{G}_1, g$ arbitrary $\Rightarrow f \vee g \in \tilde{G}_1$ and $\bar{f} \wedge g \in \tilde{G}_0$.

The sets that can be constructed from (1) using (2) and (3) are

$$\tilde{G}_0 = \left\{ \bigwedge_{i=1}^{m} \bar{c}_i f_i \mid f_i : \mathbb{B}^n \to \mathbb{B} \right\} \tag{25}$$

$$\tilde{G}_1 = \{1 \oplus f \mid f \in G_0\} \ . \tag{26}$$

These are stable under (2), since $f \vee g = 1 \oplus \bar{f}\bar{g} = f \oplus g \oplus fg$ and $f = 1 \oplus \bar{f}$. Further, the set of Boolean functions used in (2) is complete.

Appendix D
Multi-key Chains

In some situations, the proof of honest encryption method laid out in Section 3.4 might not be favourable, for example if the feature of (pseudo) random ciphers is crucial or re-computing the encryption on the receiver side is to costly. In such cases, we can still substantially weaken the oracle attack. The key idea here is to introduce redundancy, which increases key length and all run times by a constant factor.

The private/public key chain now consists of $\gamma \in \mathbb{N}$ different private/public key pairs. To encrypt a message, each bit is to be encrypted with each of the public keys, i.e. the cipher for one bit now consists of γ many Boolean functions in ANF. In a valid cipher, all of these functions evaluate to the same value upon inserting the respective private keys.

After decoding, the recipient has to decide whether or not to accept the message (bit), if the cipher is invalid. Rejecting all invalid ciphers would reveal as much information about the private key as the simple version of Section 2. Therefore we fix a threshold $t \in \{2, \dots, \gamma/2\}$ and accept the bit with value given by the majority, if the minority is smaller than t. To properly choose t, we consider the two possible outcomes of the attack:

1) If the message is rejected, the attacker learns that more than the (known) threshold t of the bits he guessed did not match the private keys.
2) If the message is accepted, the attacker will learn whether or not he guessed the majority of bits right (from the reply of the receiver).

To keep the information leakage about the private key as small as possible, we should choose t large in order to make (1) unlikely. More precisely, if $f \in \{0. \dots, \gamma\}$ of the encoded versions of the bit are manipulated by replacing one variable with a guessed value (such that the change in the function influences its value), the probability of rejection is

$$\mathrm{prob}_r(t, f) = \frac{1}{2^f} \sum_{t \le T \le f} \binom{f}{T} \ , \tag{27}$$

which can be expressed through the error function for large f. If f is close or even equal to t, the attacker gains substantial information, but for this attack, the success probability is exponentially small in t.

In the opposite case of choosing f close to γ, the attack is most likely detected if t is small enough. Concretely choosing e.g.

$$t = \frac{\gamma}{2} - \frac{c}{2}\sqrt{\gamma} \tag{28}$$

with $c = 3$ means that more than 99.7% (for large γ) of the attacks will be detected with hardly any information gain for the attacker. More precisely, the probability of a successfully attack of this type scales down super exponentially (with the complementary error function) and is below 10^{-8} already for $c = 6$. In practice, choosing $t \propto \gamma$ with a proportionality factor slightly smaller than $1/2$ should be most reasonable.

The attacker gains most information from an attack with $\mathrm{prob}_r(t, f) \approx 1/2$, i.e. $f \approx 2t$. This attack being rejected or not indicates that more or less than half of the private key bits were guessed correctly. Gathering this knowledge from $O(\gamma^2)$ attacks in a suitable scheme will reveal the value of all γ bits of the keys with a high confidence level on the attackers side. Hence the multi-key hardened version is not fully resistant against the oracle attack, but the attack will almost surely be noticed before substantial information is leaked. This invalidates the authenticity of the attacker in an authenticated communication. Even if information was leaked, not all keys in the chain have to be replaced, which limits the damage of a successful attack.

One can improve the scheme a little more by also choosing γ at random for each decryption, but still statistical analysis will eventually reveal the private key bits. Hence the proof of honest encryption scheme introduced in Section 3.4.1 is certainly more secure than the multi-key scheme.

Appendix E
Benchmarks

We have conducted some simple bench marks for breaking the public key, i.e. solving the SAT instance, using MiniSat [3]. It is well known that random instances are the hardest for $m \approx m_c = \alpha_c(k)n$. For the planted instances that we use as public keys, it turns out that the hardest instances have slightly more clauses.

E.1 3-SAT

For $k = 3$ we find (see Figure 7 to 2) that the hardest instance have $m \approx 5n$ where $m_c \approx 4.2n$ [4]. In Figure 2 we show the run time of MiniSat for various random instances generated by our key generator as a function of the number of variables for fixed $m/n = 4.3$. This is very close to the critical ratio. Fitting an exponential function to the minimal run times, we extrapolate that one should choose $n_{100} \approx 1500$ to ensure that the MiniSat solver would take at least 100 years to break the public key. For $m/n = 4.5$, as displayed in Figure 3, we find the same n_{100} within the error of our approximation, but for $m/n = 5$ (Figure 4) $n_{100} < 1000$ is significantly smaller. Increasing m/n further leads to an increases in n_{100} which again reaches $n_{100} \approx 1500$ for $m/n = 8$. Consequently, we choose $m/n = 5$ and a private key length $n = 2^{10}$ for $k = 3$ as the default values for the proof-of-concept implementation [6]. This leads to a public key length of about $k * m * (\log_2(n) + 1) \approx 165\,\text{kbit}$. Notice that the keys generated with these parameters can be considered "PC-secure" but securing the keys against more sophisticated massively parallel solvers rather requires $n \gtrsim 2^{11}$.

E.2 4-SAT

For $k = 4$ the critical ratio of clauses to variables is about $m_c = 9.8n$. Using $m/n = 10$ our MiniSat benchmarks indicate $n_{100} \approx 350$. So the private key size can be reduced significantly by using higher k. However, the public key size for these parameters ($\approx 133\,\text{kbit}$) is comparable to the one for $k = 3$. This means that the run time for encryption with higher k is significantly longer (exponential in k, see Section 2.2). Overall, it does not pay off to use higher values of k.

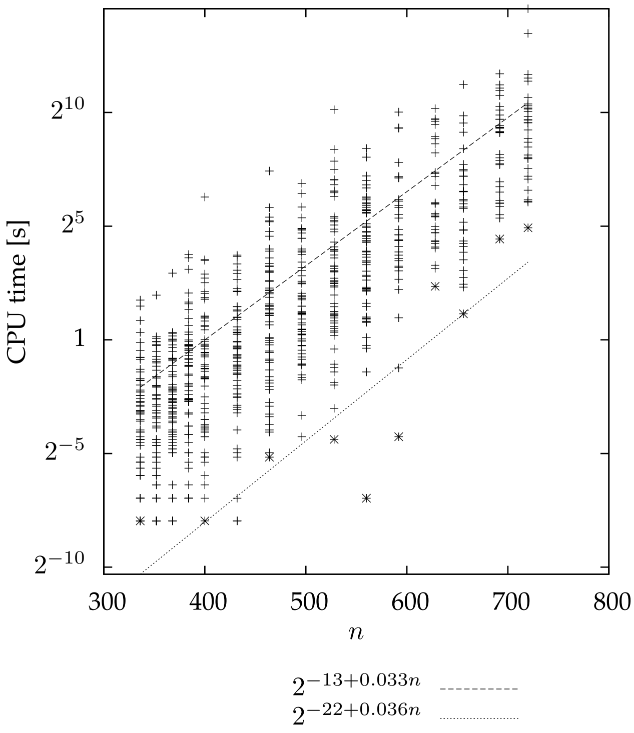

Figure 2. MiniSat run times for $m/n = 4.3$.

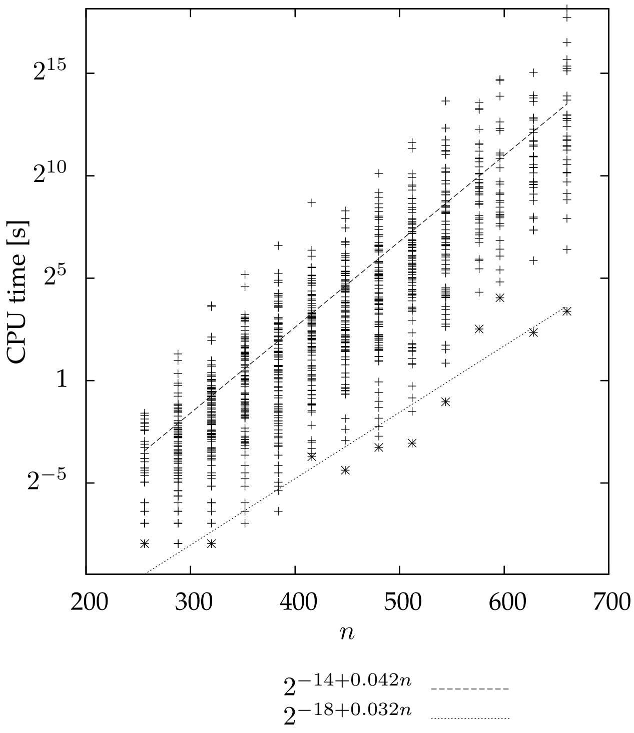

Figure 3. MiniSat run times for $m/n = 4.5$.

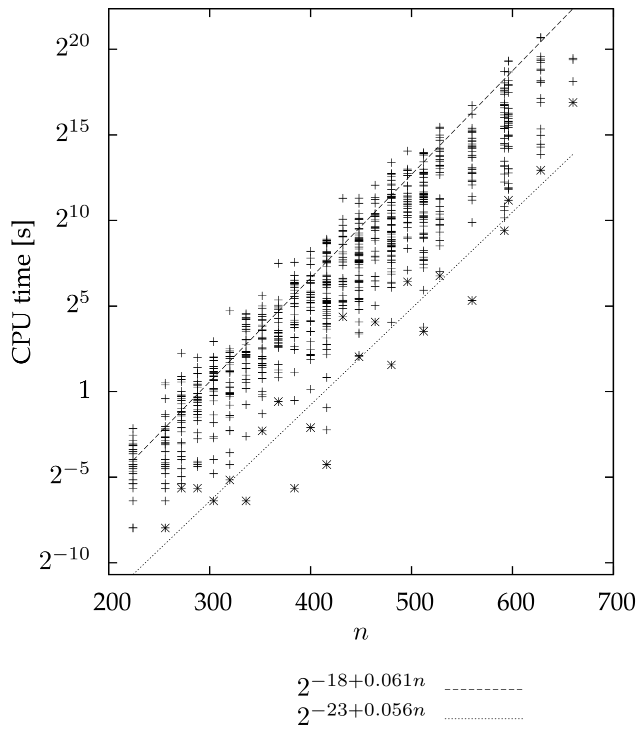

Figure 4. MiniSat run times for $m/n = 5$.

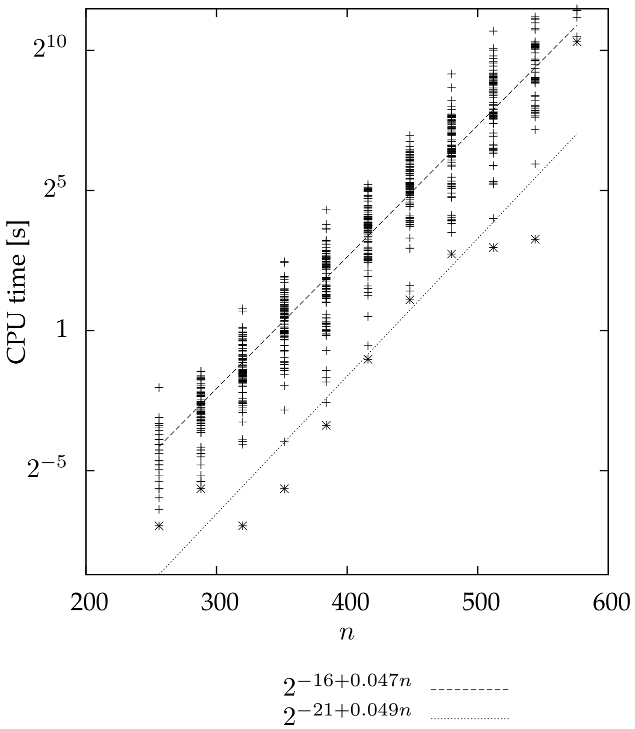

Figure 5. MiniSat run times for $m/n = 6$.

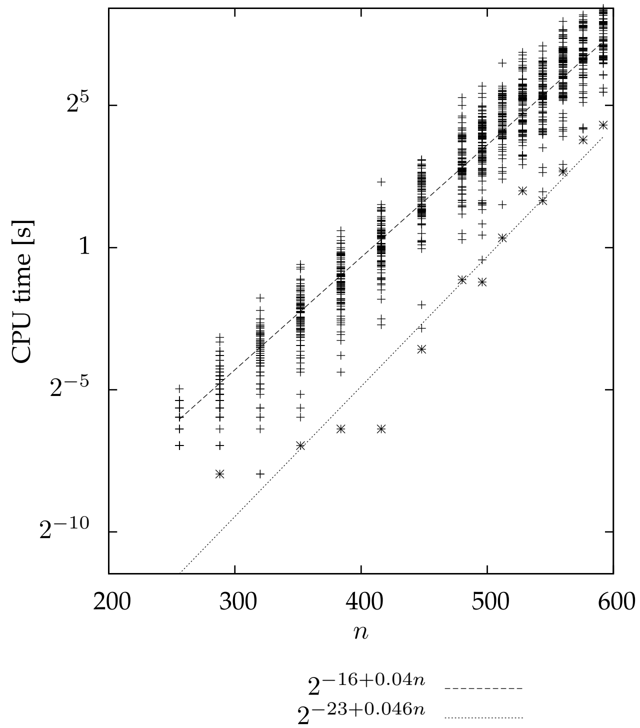

Figure 6. MiniSat run times for $m/n = 7$.

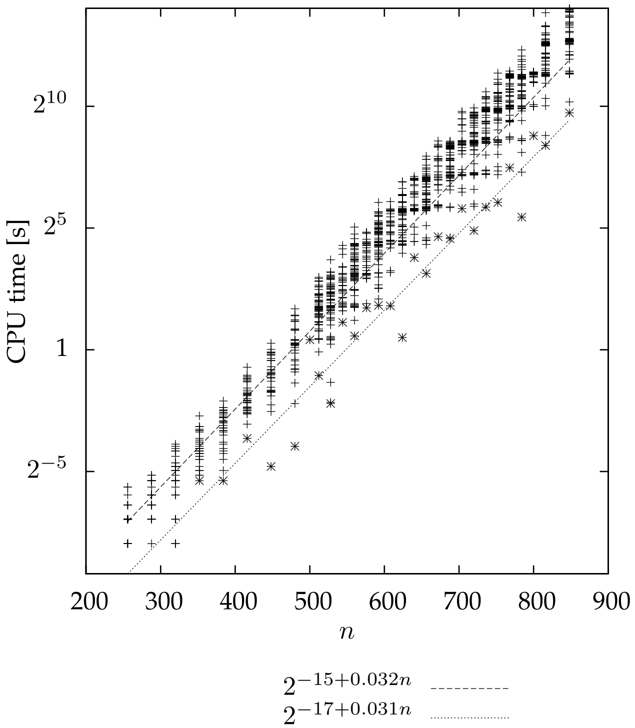

Figure 7. MiniSat run times for $m/n = 8$.