Chapter 5 The New Attack on RSA Cryptosystem

5.1 Cryptanalysis of RSA

Dan Boneh and Glenn Durfee (Computer Science Department, Stanford University, Stanford, USA) showed that if the private exponent \( d \) used in the RSA public-key cryptosystem is less than \( N^{0.292} \) then the system is insecure (D. Boneh and G. Durfee, 2000). This is the first improvement over an old result of Wiener showing that when \( d < N^{0.25} \) the RSA system is insecure (M.J. Weiner, 1990). We hope our approach can be used to eventually improve the bound to \( d < N^{0.5} \).

To provide fast RSA signature generation one is tempted to use a small private exponent \( d \). Unfortunately, Wiener showed over ten years ago that if one uses \( d < N^{0.25} \) then the RSA system can be broken. Since then there have been no improvements to this bound. Verheul and Tilborg (Verheul and Tilborg springer-verlag, 1997) showed that as long as \( d < N^{0.5} \) it is possible to expose \( d \) in less time than an exhaustive search; however, their algorithm requires exponential time as soon as \( d > N^{0.25} \). In thesis we gave the substantial improvement to Wiener's result. We show that as long as \( d < N^{0.292} \) one can efficiently break the system. We hope our approach will eventually lead to what we believe is the correct bound, namely \( d < N^{0.5} \). Our results are based on the seminal work of Coppersmith Wiener describes a number of clever techniques for avoiding his attack while still providing fast RSA signature generation. One such suggestion is to use a large value of \( e \). Indeed, Wiener's attack provides no information as soon as \( e > N^{1.5} \). In contrast, our approach is effective as long as \( e < N^{1.875} \). Consequently, larger values of \( e \) must be used to defeat the attack.

5.2 Overview of Our Approach

Recall that an RSA public key is a pair \( (N, e) \) where \( N = p.q \) is the product of two \( n \)-bit primes. For simplicity, we assume \( \gcd(p-1, q-1) = 2 \). The corresponding private key is a pair \( (N, d) \) where \( e.d \equiv 1 \mod \varphi(N) \) where \( \varphi(N) = N - p - q + 1 \)

Note that both \( e \) and \( d \) are less than \( \varphi(N) \). It follows that there exists an integer \( k \) such that
We have taken $e = N^α$ for same $α$ & $e$ is of the same order of magnitude as $N$ (e.g. $e > N / 10$) and therefore $α$ is very close to 1. As we shall see, when $α$ is much smaller than 1 our results become even stronger. Suppose the private exponent $d$ satisfies $d < N^δ$. Wiener’s results show that when $δ < 0.25$ the value of $d$ can be efficiently found given $e$ and $N$. Our goal is to show that the same holds for larger values of $δ$. By equation (1) we know that

\[ |k| < \frac{2de}{\phi(N)} \leq 3de/N < 3e^{1+\frac{δ-1}{α}}. \]

Similarly, we know that

\[ |s| < 2N^{0.5} = 2e^{1/2α}. \]

To summarize, taking $α \approx 1$ (which is the common case) and ignoring constants, we end up with the following problem: find integer’s $k$ and $s$ satisfying

\[ k(A + s) \equiv 1 \pmod{e}. \]

where $|s| < e^{0.5}$ and $|k| < e^δ$.

The problem can be viewed as follows: given an integer $A$, find an element “close” to $A$ whose inverse modulo $e$ is “small”. We refer to this is the small inverse problem. Clearly, if for a given value of $δ < 0.5$ one can efficiently list all the solutions to the small inverse problem, then RSA with private exponent smaller than $N^δ$ is insecure (simply observe that given $s$ modulo $e$ one can factor $N$ immediately, since $e > s$). Currently we can solve the small inverse problem whenever $δ < 1 – 1/2 \sqrt{2} \approx 0.292$

### 5.3 SAT Verification Methods

Automatic verification of hardware and software implementations is crucial for building reliable computer systems. Most verification tools rely on decision
procedures to check the satisfiability of various formulas that are generated during the verification process. This thesis develops new techniques for building efficient decision procedures and adds new capabilities to the existing decision procedures for certain logics.

Boolean satisfiability (SAT) solvers are used heavily in verification tools as decision procedures for propositional logic. Most state-of-the-art SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm (Davis, M., Putnam, H., 1960) and require the input formula to be in Conjunctive Normal Form (CNF). However, typical formulas that arise in practice are non-clausal, that is, not in CNF. Converting a general formula to CNF introduces overhead in the form of new variables and may destroy the structure of the initial formula, which can be useful to check satisfiability efficiently. We present two non-clausal SAT algorithms that operate on the Negation Normal Form (NNF) of the given formula. The NNF of a formula is usually more succinct than the CNF of the formula. The first algorithm is based on the idea of General Matings developed by Andrews in 1981. We develop techniques for performing search space pruning, learning, non-chronological backtracking in the context of a General Matings based SAT solver. The second algorithm applies the DPLL algorithm to NNF formulas. We devise new algorithms for performing Boolean Constraint Propagation (BCP), a key task in the DPLL algorithm.

Most hardware verification tools convert a high level design into a low level representation called a netlist for verification. However, algorithms that operate at the netlist level are unable to exploit the structure of the higher abstraction levels such as register transfer level, and thus, are less scalable. This thesis proposes the use of predicate abstraction for verifying register transfer level (RTL) Verilog. Predicate abstraction is a technique introduced for software verification.

There are two challenges when applying predicate abstraction to circuits:

(i) The computation of the abstract model in the presence of a large number of predicates.

(ii) Discovery of suitable word-level predicates for abstraction refinement.

We address the first problem using a technique called predicate clustering. We address the second problem by computing weakest pre-conditions of Verilog
statements in order to obtain new word-level predicates during abstraction refinement. An alternative technique for finding new predicates for refinement is based on the computation of Craig interpolants. Efficient algorithms are known for computing interpolants in rational and real linear arithmetic. We focus on subsets of integer linear arithmetic. Our main results are polynomial time algorithms for obtaining proofs of unsatisfiability and interpolants for conjunctions of linear diophantine equations, linear modular equations (linear congruences), and linear diophantine disequations. We show the utility of our interpolation algorithms for discovering modular/divisibility predicates in a counterexample guided abstraction refinement (CEGAR) framework. This has enabled verification of simple programs that cannot be checked using existing CEGAR based model checkers.

In complexity theory, the satisfiability problem (SAT) is a decision problem, whose instance is a Boolean expression written using only AND, OR, NOT, variables, and parentheses. The question is: given the expression, is there some assignment of TRUE and FALSE values to the variables that will make the entire expression true? A formula of propositional logic is said to be satisfiable if logical values can be assigned to its variables in a way that makes the formula true. The boolean satisfiability problem is NP-complete. The propositional satisfiability problem (PSAT), which decides whether a given propositional formula is satisfiable, is of central importance in various areas of computer science, including theoretical computer science, algorithmic, artificial intelligence, design, electronic, and verification.

A literal is either a variable or the negation of a variable (the negation of an expression can be reduced to negated variables by De Morgan’s laws). For example, \(x_1\) is a positive literal and \(\neg(x_2)\) is a negative literal. A clause is a disjunction of literals.

For example, \(x_1 \lor \neg(x_2)\) is a clause?

There are several special cases of the Boolean satisfiability problem in which the formulae are required to be conjunctions of clauses (i.e. formulae in conjunctive normal form). Determining the satisfiability of a formula in conjunctive normal form where each clause is limited to at most three literals is NP-complete; this problem is called "3SAT", "3CNFSAT", or "3-satisfiability". Determining the satisfiability of a
formula in which each clause is limited to at most two literals is NL-complete; this problem is called "2SAT". Determining the satisfiability of a formula in which each clause is a Horn clause (i.e. it contains at most one positive literal) is P-complete; this problem is called Horn-satisfiability.

The Cook–Levin theorem proves that the Boolean satisfiability problem is NP-complete, and in fact, this was the first decision problem proved to be NP-complete. However, beyond this theoretical significance, efficient and scalable algorithms for SAT that were developed over the last decade have contributed to dramatic advances in our ability to automatically solve problem instances involving tens of thousands of variables and millions of constraints. Examples of such problems in electronic design automation (EDA) include formal equivalence checking, model checking, formal verification of pipelined microprocessors, automatic test pattern generation, routing of FPGAs, and so on. A SAT-solving engine is now considered to be an essential component in the EDA toolbox.

5.4 NP-completeness

SAT was the first known NP-complete problem, as proved by Stephen Cook in 1971. Until that time, the concept of an NP-complete problem did not even exist. The problem remains NP-complete even if all expressions are written in conjunctive normal form with 3 variables per clause (3-CNF), yielding the 3SAT problem. This means the expression has the form:

\[(x_{11} \lor x_{12} \lor x_{13}) \land (x_{21} \lor x_{22} \lor x_{23}) \land (x_{31} \lor x_{32} \lor x_{33}) \land \ldots \]

where each \(x\) is a variable or a negation of a variable, and each variable can appear multiple times in the expression. A useful property of Cook's reduction is that it preserves the number of accepting answers. For example, if a graph has 17 valid 3-colorings, the SAT formula produced by the reduction will have 17 satisfying assignments. NP-completeness only refers to the run-time of the worst case instances. Many of the instances that occur in practical applications can be solved much more quickly. See runtime behaviour below. SAT is easier if the formulas are restricted to those in disjunctive normal form, that is, they are disjunction (OR) of terms, where
each term is a conjunction (AND) of literals (possibly negated variables). Such a formula is indeed satisfiable if and only if at least one of its terms is satisfiable, and a term is satisfiable if and only if it does not contain both x and NOT x for some variable x. This can be checked in polynomial time.

5.4.1 2-satisfiability

SAT is also easier if the number of literals in a clause is limited to 2, in which case the problem is called 2SAT. This problem can also be solved in polynomial time, and in fact is complete for the class NL. Similarly, if we limit the number of literals per clause to 2 and change the AND operations to XOR operations, the result is exclusive-or 2-satisfiability, a problem complete for SL = L.

One of the most important restrictions of SAT is HORNSAT, where the formula is a conjunction of Horn clauses. This problem is solved by the polynomial-time Horn-satisfiability algorithm, and is in fact P-complete. It can be seen as P's version of the Boolean satisfiability problem.

Provided that the complexity classes P and NP are not equal, none of these restrictions are NP-complete, unlike SAT. The assumption that P and NP are not equal is currently not proven.

5.4.2 3-satisfiability

Three-satisfiability is a special case of k-satisfiability (k-SAT) or simply satisfiability (SAT), when each clause contains exactly k = 3 literals. It was one of Karp's 21 NP-complete problems. Here is an example, where ¬ indicates negation:

E = (x_1 or ¬x_2 or ¬x_3) and (x_1 or x_2 or x_4)

E has two clauses (denoted by parentheses), four variables (x_1, x_2, x_3, x_4), and k=3 (three literals per clause). To solve this instance of the decision problem we must determine whether there is a truth value (TRUE or FALSE) we can assign to each of the literals (x_1 through x_4) such that the entire expression is TRUE. In this instance, there is such an assignment (x_1 = TRUE, x_2 = TRUE, x_3=TRUE, x_4=TRUE), so the answer to this instance is YES. This is one of many possible assignments, with for instance, any set of assignments including x_1 = TRUE being sufficient. If there were no such assignment(s), the answer would be number.
Since k-SAT (the general case) reduces to 3-SAT, and 3-SAT can be proven to be NP-complete, it can be used to prove that other problems are also NP-complete. This is done by showing how a solution to another problem could be used to solve 3-SAT. An example of a problem where this method has been used is "Clique". It’s often easier to use reductions from 3-SAT than SAT to problems that researchers are attempting to prove NP-complete.

3-SAT can be further restricted to One-in-three 3SAT, where we ask if exactly one of the variables in each clause is true, rather than at least one. One-in-three 3SAT remains NP-complete.

Algorithms for solving SAT-There are two classes of high-performance algorithms for solving instances of SAT in practice: modern variants of the DPLL algorithm, such as Chaff, GRASP or march, and stochastic local search algorithms, such as WalkSAT. A DPLL SAT solver employs a systematic backtracking search procedure to explore the (exponentially-sized) space of variable assignments looking for satisfying assignments. The basic search procedure was proposed in two seminal papers in the early 60s and is now commonly referred to as the Davis-Putnam-Logemann-Loveland algorithm ("DPLL" or "DLL"). Theoretically, exponential lower bounds have been proved for the DPLL family of algorithms.

5.5 Algorithms for Solving SAT

Modern SAT solvers (developed in the last ten years) come in two classes: "conflict-driven" and "look-ahead". Conflict-driven solvers augment the basic DPLL search algorithm with efficient conflict analysis, clause learning, non-chronological backtracking (aka back jumping), as well as "two-watched-literals" unit propagation, adaptive branching, and random restarts. These "extras" to the basic systematic search have been empirically shown to be essential for handling the large SAT instances that arise in Electronic Design Automation (EDA). Look-ahead solvers have especially strengthened reductions (going beyond unit-clause propagation) and the heuristics, and they are generally stronger than conflict-driven solvers on hard instances (while conflict-driven solvers can be much better on large instances which have inside actually an easy instance).
Modern SAT solvers are also having significant impact on the fields of software verification, constraint solving in artificial intelligence, and operations research, among others. Powerful solvers are readily available as free and open source software. In particular, the conflict-driven MiniSAT, which was relatively successful at the 2005 SAT competition, only has about 600 lines of code. An example for look-ahead solvers is march_dl, which won a prize at the 2007 SAT competition.

Genetic algorithms and other general-purpose stochastic local search methods are also being used to solve SAT problems, especially when there is no or limited knowledge of the specific structure of the problem instances to be solved. Certain types of large random satisfiable instances of SAT can be solved by survey propagation (SP). Particularly in hardware design and verification applications, satisfiability and other logical properties of a given propositional formula are sometimes decided based on a representation of the formula as a binary decision diagram (BDD).

Propositional satisfiability has various generalisations, including satisfiability for quantified Boolean formula problem, for first- and second-order logic, constraint satisfaction problems, 0-1 integer programming, and maximum satisfiability problem. Many other decision problems, such as graph coloring problems, planning problems, and scheduling problems, can be easily encoded into S.

5.6 Conclusion

In this chapter we are analysing the different results given by Dan Boneh, Glenn Durfee & Weiner about increasing the range of the private exponent (d). We hope our approach can be used to eventually improve the bound to \( d < N^{0.5} \). The use of satisfiability problem (SAT) it is a decision problem, whose instance is a Boolean expression written using only AND, OR, NOT, variables, and parentheses. NP-Completeness problem is mainly based on SAT solver; different types of satisfiability are available like 2-satisfiability, 3-satisfiability & Horn satisfiability which are based on runtime polynomials depending upon the number of clauses in each term.

Modern SAT solver is basically design on the concept DPLL search algorithm. It also has significant impact on the fields of software verification, constraint solving in artificial intelligence, and operations research, among others. Genetic
algorithms and other general-purpose stochastic local search methods are also being used to solve SAT problems. As we have seen that in SAT competition of 2005, 2007 and many more SAT has been successful in solving problems efficiently.