Generation of Strong Primes

Cryptography and Computer Security Project

Author: Samo Metličar
Supervisor: Aleksandar Jurišić

September 5, 2020

Contents

Abstract

In this work strong primes and the incentives to use them in cryptography are described. An algorithm presented by J. Gordon in 1985 is studied and implemented with slight modifications. Prime numbers generated with said algorithm are tested for compatibility with RSA cryptosystem.

Key words

Strong primes, algorithm, RSA, time complexity, generation, bit length.

1 Introduction

Prime numbers are an essential part of cryptography and their properties are the building blocks of many cryptosystems. One such system is RSA[1] [8], cf. Stinson and Paterson [9]. Most attacks on RSA try to factor $n = pq$ and $\varphi(n) = (p-1)(q-1)$, where $p, q \in \mathbb{P}$. Failing to protect the system against such attacks could have negative consequences, e.g. an attacker may even obtain private keys. Therefore, it is of great importance to increase security against different attacks, which is why additional conditions for primes p and q were established. Primes satisfying those conditions are called strong primes and will be discussed in this project.

Our goal is to define and generate strong primes and compare them to modern RSA standards (e.g. `FIPS 186-4` [11]). We do this using a modified version of the Gordon algorithm [2].

In the paper we define strong primes and explain the benefits of added conditions in Section 2, then we construct the algorithm in Section 3 and give an overview of the implementation in Section 4. In Section 5 the distribution of generated primes is analyzed and in Section 6 strong primes are tested for compatibility with RSA.

2 Strong primes

We mentioned that several known attacks on the RSA caused additional conditions to be added to prime numbers used for its keys. The first such attack is the `Pollard p-1 algorithm` [7], cf. Stinson and Paterson [9]. It is a factoring algorithm and its time complexity depends on the largest factor of the input integer. Therefore the algorithm will be inefficient if for some $p \in \mathbb{P}$, $p - 1$ has a large enough factor.

The second attack is `Williams' p+1 algorithm` [10]. It is analogous to the previous attack. The difference is, it's the most efficient when $p + 1$ is smooth, i.e. it contains only small factors. To protect the system against such attacks the prime numbers $p, q \in \mathbb{P}$ used for RSA should be such that $p + 1$ and $q + 1$ both have a large enough factor.

The final condition increases security against `cycle attacks`, Ferrucci and Pornin [1], and Killeen [4]. Such attacks start by selecting a message and encrypting it until at some step the plaintext is obtained again, while also keeping track of the number of iterations. From this we can obtain the private key. Described attack will always work but it is also inefficient for modern key bit lengths. The algorithm can be additionally slowed down by choosing such primes p and q for RSA, that for large factors p_1 and q_1 of $p - 1$ and $q - 1$ respectively, $p_1 - 1$ and $q_1 - 1$ also have large factors.

2.1 Definition of strong primes

An integer $p \in \mathbb{N}$ is a strong prime if it satisfies the following conditions:

[1] Rivest-Shamir-Adleman is one of the first public-key cryptosystems and has been used for over 40 years.

- p is prime;
- p is sufficiently large to be used in cryptography;
- $p-1$ has a large prime factor, say r;
- $p+1$ has a large prime factor, say s;
- $r-1$ has a large prime factor, say t.

If $p-1$ has a large prime factor r, then $p = Kr+1$ for some $K \in \mathbb{N}$. Because p and r are large primes, therefore not 2, they are odd and K must be even. A strong prime can therefore only be a succifiently large odd prime with the following properties:

1. $p \equiv 1 \pmod{2r}$,
2. $p \equiv -1 \pmod{2s}$,
3. $r \equiv 1 \pmod{2t}$,

where r, s and t are large primes. We will use this notation for the rest of the project, unless stated otherwise.

3 Algorithm

The algorithm requires a random number generator, which will not be described in the project and it will be assumed that we can generate random integers of desired bit length. A strong prime will be constructed in four steps:

1. choose random seeds a and b,
2. from a and b generate random primes s and t, where $s > a$ and $t > b$,
3. from t construct r,
4. from r and s construct strong prime p.

3.1 Generating s and t

Say we have random seeds a and b. Primes s and t will be the first prime numbers greater than a and b respectively. For s we therefore traverse through integers larger than a, starting at $a+1$ and check for primality at each step. The latter can be done using probabilistic algorithms such as `Solovay-Strassen Algorithm` and `Miller-Rabin Algorithm`, Stinson and Paterson [9]. We repeat the steps for t, this time starting at $b+1$.
Additionally we can only traverse through integers that are not divisible by the first k prime numbers for some $k \in \mathbb{N}$ (e.g. first 54 primes, i.e. all 8-bit primes). By doing this we only check for primality on $\prod_{i=1}^{54}(1-\frac{1}{p_i}) \approx 0.10035$ of all integers,[2] where p_i is the i-th prime number.

We can ensure that s and t will have the same number of bits as a and b, say n, by picking random seeds in the range $[2^{n-1}, 2^{n-1}+2^{n-2})$. This leaves us at least $2^n - (2^{n-1}+2^{n-2}) = 2^{n-2}$ integers in which to find prime numbers s and t before bit length would increase.

[2]This is an approximation, since there is potentially some overlap. In practice it turns out to be accurate enough.

3.2 Constructing r from t

Because the prime r satisfies condition 3 in Section 2.1, we know that it will be an element of $\{2Lt+1\}_{L=1}^{\infty}$. We could find the smallest L for which $r = 2Lt+1$ is prime, but we would have difficulties controlling the bit length of r, since every time L doubles, r gains a bit.
We therefore wish to find bounds L_{low} and L_{high} that for every $\ell \in [L_{\text{low}}, L_{\text{high}})$ the value of $2\ell t+1$ is of the desired bit length. The first $\ell > L_{\text{low}}$, s.t. $r = 2\ell t+1$ is prime, is then chosen. If the size of t is chosen appropriately, the size of the interval from L_{low} to L_{high} is large enough for the probability of not finding such prime r of desired bit length to be nearly zero. More on this in Section 4.1, including the procedure of finding L_{low} and L_{high}.

Similar conclusions can be made about the distribution of prime numbers r as in Section 5.

3.3 Constructing p from r and s

Because the prime p follows conditions 1 and 2 in Section 2.1, it will be an element of both $\{2Kr+1\}_{K=1}^{\infty}$ and $\{2Ls-1\}_{L=1}^{\infty}$. In other words $p = 2kr+1 = 2\ell s - 1$ for some $k, \ell \in \mathbb{N}$. We will find such p using the following theorem.

Theorem 1. *If r and s are odd primes, then p satisfies:*

$$\begin{aligned} p &\equiv 1 \pmod{2r} \\ p &\equiv -1 \pmod{2s}, \end{aligned} \tag{1}$$

if and only if p is of the form

$$p = p_0 + 2krs, \tag{2}$$

where $k \in \mathbb{N}$,

$$p_0 = \begin{cases} u(r,s) & \text{if } u(r,s) \text{ is odd,} \\ u(r,s) + rs & \text{if } u(r,s) \text{ is even,} \end{cases} \tag{3}$$

and $u(r,s) = (s^{r-1} - r^{s-1}) \bmod rs$.

Proof. Let $p \in \mathbb{N}$ be any integer satisfying (1). Because $p-1$ is divisible by $2r$, it means it's also divisible by r. Same goes for $p+1$ and $2s$. We then get a weaker condition

$$p = jr + 1 = \ell s - 1 \quad \text{for some } j, \ell \in \mathbb{N}. \tag{4}$$

We observe that for even j, ℓ we get the condition (1). This means that odd numbers satisfying (4) also satisfy (1). We therefore want to prove that solutions of (4) are of the form $u(r,s) + krs$ for some $k \in \mathbb{N}$.

Using `Fermat's Little Theorem`, Stinson and Paterson [9], we get the following congruences:

$$\begin{aligned} s^{r-1} &\equiv 1 \pmod{r}, \\ r^{s-1} &\equiv 1 \pmod{s}. \end{aligned}$$

Trivially, it also holds that $s^{r-1} \equiv 0 \pmod{s}$, $r^{s-1} \equiv 0 \pmod{r}$, $rs \equiv 0 \pmod{s}$ and $rs \equiv 0 \pmod{r}$. From this it follows that $u(r, s)$ satisfies (4).

We now show that all solutions of (4) are of the form $u(r, s) + krs$. We prove this by choosing $u = jr + 1 = \ell s - 1$ and $u' = j'r + 1 = \ell' s - 1$ for some $j, j', \ell, \ell' \in \mathbb{N}$, this means that u and u' satisfy (4). We observe $u \equiv u' \equiv 1 \pmod{r}$ and $u \equiv u' \equiv -1 \pmod{s}$. So

$$\begin{aligned} u - u' &= (1 \bmod r) - (1 \bmod r) \\ &= 0 \bmod r \\ &= kr, \\ u - u' &= (-1 \bmod s) - (-1 \bmod s) \\ &= 0 \bmod s \\ &= k's, \end{aligned} \tag{5}$$

for some integers k, k'. From this it follows that $u - u'$ is a multiple of $\operatorname{lcm}(r, s) = rs$, since r and s are prime. Because $u(r, s)$ satisfies (4), u and u' must be of the form $u(r, s) + ksr$. □

We now know that p will be an element of $\{p_0 + 2Krs\}_{K=1}^{\infty}$, where p_0 was defined in Theorem 1. The same issues with bit sizes may arrise as with constructing r. We thus use the same method of finding an interval $[K_{\text{low}}, K_{\text{high}})$, where K_{low} can be adjusted to be the first integer such that $p_0 + K_{\text{low}}rs \geq \sqrt{2} \cdot 2^{n-1}$ to comply with `FIPS PUB 186-4`, where n is the desired bit length.

It turns out for this to be achievable for a strong n-bit prime p, primes r, s and t should roughly have $(n/2)$-bits each. Sizes of r and s should also be equal and while lowering them increases the probability of finding a strong prime p with n bits, it also lowers its security. Controlling the bit length of s does not present any problems and we have shown how to construct r of desired size. Exact values are calculated in Section 4.1.

3.4 Time complexity

Denote with $T_{\text{prime}}(n)$ the time needed to check if a n-bit integer is prime with a method of choice, and with $T_{\text{exp}}(n)$ the time complexity of exponentiation.

The time complexity of step 3.1 depends on how much preprocessing is done, i.e. calculating primes to be used for skipping integers that are not potential primes. Say that we only consider $P \in (0, 1]$ of integers. Using `Prime number theory`, Stinson and Paterson [9], we need to check for primality for $P \ln(x)$ integers.[3] For a n-bit prime number, this adds to $P \ln(2^n) = nP \ln(2) \approx 0.7Pn$. For each of those $nP \ln(2)$ integers we use a primality test with time complexity $T_{\text{prime}}(n)$, therefore the first step costs $\frac{nP}{2} \ln(2) T_{\text{prime}}(\frac{n}{2})$ operations for each prime.

Step 3.2 is similar to Step 3.1 as far as the time complexity is concerned, since we are checking roughly the same number of integers.

The final step first requires two exponentiations, each with time complexity $T_{\text{exp}}(n)$, which is dominated by the time spent searching for primes. Additionally, we need to find a prime, which takes $nP \ln(2) T_{\text{prime}}(n)$ operations.

[3] Probability of a prime in vicinity of x is estimated with $\frac{\mathrm{d}}{\mathrm{d}x} \frac{x}{\ln x} \approx \frac{1}{\ln x}$ for large enough x.

The algorithm searches for a n-bit prime p, and for roughly $\frac{n}{2}$-bit primes t, r and s. Together then, ignoring everything but time spent searching for primes, we obtain

$$nP\ln(2)T_{\text{prime}}(n) + 3\frac{n}{2}P\ln(2)T_{\text{prime}}(\frac{n}{2}). \tag{6}$$

If we now assume that a probabilistic algorithm for primality testing is used (e.g. `Miller-Rabin algorithm`), the time complexity is of order 3. This means that the total time complexity equals $\frac{19}{16}nP\ln(2)T_{\text{prime}}(n)$, which is $\frac{3}{16} = 0.1875$ cost to performance compared to searching for a random n-bit prime.

4 Implementation of the algorithm

Our algorithm, see [5], takes the following inputs:

- n - desired number of bits for a strong prime,
- rand - a function that takes two integers m_0 and m_1 and returns a random integer in the range $[m_0, m_1)$,
- rep_1 - an integer greater than 0 indicating the number of iterations of `Miller-Rabin Algorithm` for primality testing of r, s and t,
- rep_2 - an integer greater than 0 indicating the number of iterations of `Miller-Rabin Algorithm` for primality testing of p.

Output consists of prime numbers p, r, s and t described in Section 2.1.

4.1 Bit sizes

The algorithm starts by calculating the bit lengths of all prime numbers that will be constructed. We can do this in the reverse order, that is done by first working out what the bit size of r and s has to be so that constructing p will be possible in $1 - \varepsilon$ cases for some error ε. The algorithm was implemented with ε of at most 2^{-80} to comply with `FIPS PUB 186-4`. Below is the description of how this is achieved.

4.1.1 Determining bit size of t

Let us denote with n_1 the number of bits of prime number r defined in Section 2.1. Using `Prime Number Theorem`, Stinson and Paterson [9], we know that the number of primes smaller than x is approximately $\frac{x}{\ln x}$. Using this we can estimate the number of n_1 bit primes to be

$$\frac{2^{n_1}}{n_1\ln 2} - \frac{2^{n_1-1}}{(n_1-1)\ln 2} = \frac{n_1-2}{n_1(n_1-1)\ln 2}2^{n_1-1},$$

which means that the probability of randomly choosing a prime is

$$P := P[x \text{ is prime}] = \frac{n_1-2}{n_1(n_1-1)\ln 2}.$$

The probability of choosing a composite number is therefore $1 - P$. We want to know the number of integers that need to be tried for the probability of not finding a prime to be less than ε, which can be determined from $(1 - P)^k < \varepsilon$. The latter is equivalent to $k > \log_{1-p} \varepsilon$, where k is the number of attempts. It turns out that this condition is always fulfilled for $\varepsilon = 2^{-80}$ if $k \geq 2^{\lceil \log_2 n_1 \rceil + 6}$. Furthermore, the probability of failure for such k is at most 2^{-101} for $n_1 \leq 128$, 2^{-124} for $n_1 > 128$ and 2^{-133} for $n_1 > 256$. Because we are traversing through n_1-bit integers with a step $2t$ and we want at least $2^{\lceil \log_2 n_1 \rceil + 6}$ steps, prime number t can be at most $(n_1 - \lceil \log_2 n_1 \rceil - 7)$ bits long.

4.1.2 Determining bit size of r and s

We mentioned that we want r and s to be of equal bit size. Let n_1 still be the bit length of primes r and s and let n be the number of bits of p. We're interested in the relation between n and n_1. Similarly to 4.1.1, we use `Prime Number Theorem`. The difference is, we are now looking for a n-bit prime that is greater than $\sqrt{2} \cdot 2^{n-1}$. There are $\frac{2^n}{n \ln 2} - \frac{2^{n-\frac{1}{2}}}{(n-\frac{1}{2}) \ln 2}$ such primes and the probability of finding a prime is $P = \frac{1}{(n-\frac{1}{2}) \ln 2} - \frac{\sqrt{2}}{n(2n-1)(\sqrt{2}-1) \ln 2}$. Because $2rs$ can be $2n_1$ or $2n_1 + 1$, we use the following formula $n_1 = \lfloor \frac{n - \lceil \log_2 n \rceil}{2} \rfloor - 4$ to obtain the error probability of at most 2^{-133} for $n \geq 512$. Values for the most common bit sizes are shown in Figure 1.

n	n_1	n_2
512	247	232
1024	503	487

n	n_1	n_2
1536	758	741
2048	1014	997

Figure 1: Table showing the correlation between bit sizes.

4.2 Primality testing

Our implementation is using `Miller-Rabin Algorithm` for primality testing. By choosing the number of iterations we control the error probability, that is the probability that the algorithm returns `True` when its input was composite.

4.3 Preparation for the construction of r and p

In both cases, we are first adding some even number ℓ to the starting value until the desired bit length is reached ($\ell = 2t$ for r and $\ell = 2rs$ for p). This process is sped up by finding the maximum power k of 2, s.t. $2^k \ell$ is a bit shorter than the desired bit length. We then add $2^k \ell$ to the starting value until the next addition would bring it to the desired bit length. The value of k is then decreased by 1 and the procedure is repeated while $k \geq 0$. Finally, we get an integer, s.t. when we add ℓ to it, we get the smallest value of desired size and form. For generating the prime r we use $k = n_1 - n_2 - 2$ and for the prime p we use $k = n - 2n_1 - 1$.

This procedure logarithmically reduces the number of iterations needed before the algorithm can start checking for primality.

5 Density of generated primes

We observe that primes generated with the algorithm described in 3 are condensed around $2^{n-\frac{1}{2}}$, where n is the number of bits. A question may arise whether or not this can be used by attackers. Let us sketch the idea of why prime distribution has a higher density at that point and show how small of a subsection of n-bit integers we need to check as an attacker to find $1-\varepsilon$ of all keys for some $\varepsilon \in (0,1)$.

We know, that the probability of an n-bit integer greater than $2^{n-\frac{1}{2}}$ being prime is

$$p_n \approx \frac{(\sqrt{2}-1)n - \frac{\sqrt{2}}{2}}{\ln 2 \cdot (\sqrt{2}-1)n(n-\frac{1}{2})}.$$

We are now interested in how many prime candidates, i.e. number of steps in Section 3.3, we expect to try before the probability of failure, i.e. not finding a prime, is at most ε. Therefore, we are looking for a value k s.t. $(1-p_n)^k = \varepsilon$, which means that

$$k = \log_{1-p_n} \varepsilon = \frac{\ln \varepsilon}{\ln(1-p_n)} \approx -\frac{\ln \varepsilon}{p_n},$$

where we used the *Taylor series* for natural logarithms to approximate $\ln(1-p_n)$ with $-p_n$.

The last part is calculating the minimum number of steps (this will ensure we get the worst case scenario) we can take when constructing n-bit prime as described in 3.3. Let us assume that n is even and it also has an even amount of bits (similar for odd number of bits), then r and s have $n_1 = \frac{n-\log_2 n}{2} - 4$ bits each and $2rs$ has at most $2n_1 + 1 = n - \log_2 n - 7$ bits. The minimum number of steps available is therefore

$$\begin{aligned} k_{max} &= \frac{(\sqrt{2}-1)2^{n-\frac{1}{2}}}{2^{n-\log_2 n-7}} \\ &= (\sqrt{2}-1)2^{\log_2 n + \frac{13}{2}} \\ &= \sqrt{2}(\sqrt{2}-1)2^6 n. \end{aligned}$$

We now answer the initial question and obtain the subinterval of n-bit primes that we are interested in as k/k_{max}. So

$$\begin{aligned} k/k_{max} &= -\frac{\ln \varepsilon}{\sqrt{2}(\sqrt{2}-1)2^6 n p_n} \\ &= -\ln \varepsilon \frac{\ln 2 \cdot n(n-\frac{1}{2})}{\big((\sqrt{2}-1)n - \frac{\sqrt{2}}{2}\big)\sqrt{2}2^6 n}. \end{aligned}$$

Since we wish to know how this ratio behaves when n grows, we look at the limit

$$\lim_{n\to\infty} k/k_{max} = -\frac{\ln \varepsilon \cdot \ln 2}{\sqrt{2}(\sqrt{2}-1)2^6}. \tag{7}$$

The values based on ε can be seen in Figure 2.

We conclude that an attacker that would not mind missing ε keys would only need to check a small portion of the n-bit integers and that the size of that

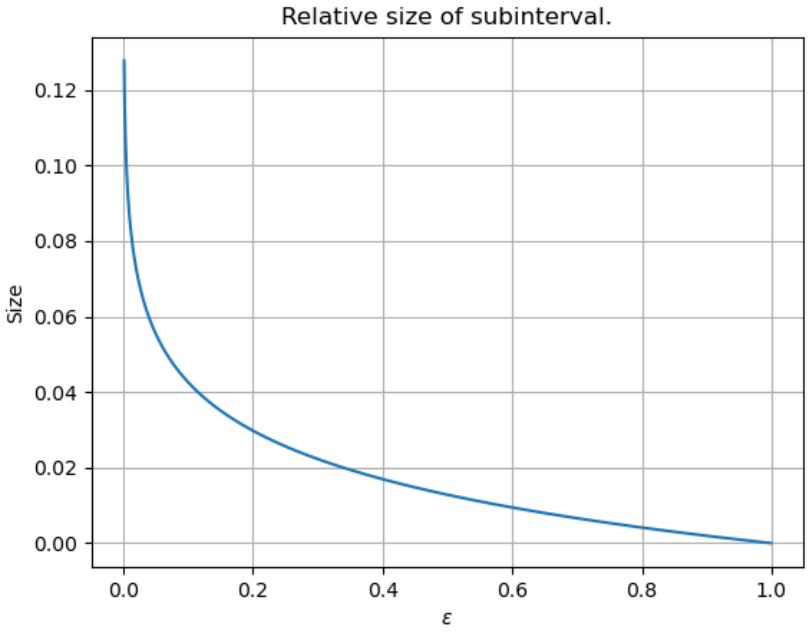

Figure 2: Size of the subinterval that contains $1 - \epsilon$ of generated primes.

subinterval is in logarithmic relation to ε. However, we showed in (7) that the limit does not converge towards 0 but rather to some constant $C_\varepsilon > 0$ (at fixed ε) and therefore does not have a noticable effect in practice. This is because we can increase the size of the key exponentially, thus also exponentially increasing the absolute size of the subinterval the attacker would have to check.

6 Compatibility with RSA

In this section our implementation will be compared to the conditions specified in `FIPS PUB 186-4`. The source requires prime numbers to satisfy a set of conditions:[4]

- integers p and q are randomly generated prime numbers,
- $(p-1)$ has a prime factor p_1,
- $(p+1)$ has a prime factor p_2,
- $(q-1)$ has a prime factor q_1,
- $(q+1)$ has a prime factor q_2,

where p_1, p_2, q_1 and q_2 are of the appropriate bit length. Primes generated with the algorithm described in Section 3 satisfy these conditions by definition and also add additional conditions.

[4]Primes satisfying the conditions are labeled as *Primes with Conditions*.

Next we look at the bit sizes of prime factors. Translating `Table B.1` from `FIPS PUB 186-4` to our notation in Figure 3, we observe that the prime numbers and their prime factors are of correct bit size.

n	min	max	$2n_1$
512	101	495	494
1024	141	1006	1006
1536	171	1517	1516

Figure 3: Maximum bit sizes of prime factors in relation to the size of primes p and q.

The last requirement regarding the process of generating a prime number is the inequality $\sqrt{2} \cdot 2^{n-1} \leq p \leq 2^n - 1$, which we took into account in Sections 3.3 and 4.1.2.

Presented algorithm should therefore comply with all of the requirements for generating keys for RSA. The code for a proof of concept can be found on `github` [5].

7 Conclusion

We have presented an algorithm for finding strong primes with relatively small loss to computing time (as shown in Section 3.4) compared to finding random primes of the same size. The algorithm was implemented in *Python* [5]. Primes generated by this method are also compatible with RSA standard [11]. However, they do have an additional condition which may be redundant in modern times and may only increase the algorithms time complexity.

References

[1] E. Ferrucci, T. Pornin. *Cycle attack on RSA, 04.01.2012, and response, 05.01.2012.* `https://crypto.stackexchange.com/a/1575`, visited 31.01.2020.

[2] J. Gordon, Cybermation Ltd. *Strong Primes are Easy to Find.* Springer-Verlag Berlin Heidelberg, 1985.

[3] J. Jonsson, B. Kaliski. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1.* RSA Laboratories, 2003.

[4] R. Killeen. *Possible Attacks on RSA: Cycle Attack.* `http://members.tripod.com/irish_ronan/rsa/attacks.html#cycle`, visited 31.01.2020.

[5] S. Metličar. *Strong Prime Generator.* `https://github.com/SamoFMF/Strong-Prime-Generator`, visited 19.08.2020.

[6] M. Nemec. *The properties of RSA key generation process in software libraries.* Faculty of Informatics, Masaryk University, 2016.

[7] J. Pollard. *Theorems on factorization and primality testing, Mathematical Proceedings of the Cambridge Philosophical Society, 76(3), pages 521-528.* Cambridge Philosophical Society, 1974.

[8] R. L. Rivest, A. Shamir, L. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Massachusetts Institute of Technology, Cambridge, 1977.

[9] D. R. Stinson, M. B. Paterson. *Cryptography: Theory and Practice - Fourth Edition.* CRC Press, Taylor & Francis Group, 2018.

[10] H.C. Williams. *A $p + 1$ Method of Factoring, Math. of Comp., Vol. 39, No. 159, pages 225-234.* American Mathematical Society, 1982.

[11] *FIPS PUB 186-4: Digital Signature Standard (DSS).* Information Technology Laboratory, National Institute of Standards and Technology, 2013.