The Korean Certificate-based Digital Signature Algorithm*

KCDSA Task Force Team†

August 10, 1998

Abstract

As a contribution to IEEE P1363a, we propose a provably secure digital signature algorithm called the Korean Certificate-based Digital Signature Algorithm (KCDSA, in short) and its elliptic curve variant (EC-KCDSA). We believe that the proposed signature schemes are more advantageous than DSA/EC-DSA in both security and efficiency. No patent related with KCDSA/EC-KCDSA has been submitted, and there are no known limitation and disadvantage. This paper describes these signature algorithms and discusses their security and efficiency aspects.

1 Introduction

The digital signature technique, a technique for signing and verifying digital documents in an unforgeable way, is essential for secure transactions over open networks. Digital signatures can be used in a variety of applications to ensure the integrity of data exchanged or stored and to prove to the recipient the originator's identity.

The security of most public key cryptosystems widely used in practice is based on two difficult problems: the problem of factoring large integers and the problem of finding discrete logarithms over finite fields. The RSA scheme [17] is designed based on the former problem and widely used in many applications as a de facto standard. On the other hand, the discrete logarithm problem is the basis of Diffie-Hellman and ElGamal-type public key systems [5, 6]. Recently, two variants of the ElGamal signature scheme have been standardized in U.S.A as digital signature standard (DSS) [20] and in Russia as GOST 34.10 (see [11]).

A group of Korean cryptographers, in association with government-supported agencies, has been developing a candidate algorithm for Korean digital signature standard, which is named KCDSA(standing for Korean Certificate-based Digital Signature Algorithm). KCDSA is a variant of ElGamal, similar to DSA and GOST, and it is designed by incorporating several features from the recent cryptographic research and thus is believed to be secure and robust. Now, KCDSA is being standardized by the Korean Government.

In this paper we describe the proposed standard for KCDSA and discuss security and efficiency aspects considered during the design process. Throughout this paper we will use the following symbols and notation:

---

*This is a modified version of the paper authored by Chae Hoon Lim (chlim@future.co.kr) and Pil Joong Lee (pjl@postech.ac.kr) which is to be presented at the ASIACRYPT'98, October 18-22, 1998.

†The team consists of Pil Joong Lee(POSTECH), Chae Hoon Lim (Future Systems, Inc.), Sang Jae Moon (Kyungpook Nat'l Univ.), Dong Ho Won (SungGyunKwan Univ.), Sung Jun Park (KISA), Chung Ryong Jang (KyungDong Univ.), Shin Gak Kang (ETRI), Eun Jeong Lee (POSTECH), Sang Bae Park (IDIS), Chul Kim (KwangWoon Univ.), Kyung Seok Lee (KIET), Jae Hyun Baek (ADD), Jong Tae Shin (KISA), Hyo Sun Hwang (Future Systems, Inc.), Sang Gyoo Sim (POSTECH), etc. The team was supported by ETRI (Electronics and Telecommunications Research Institute) and KISA (Korea Information Security Agency).

- $a \oplus b$: exclusive-or of two bit strings a and b.
- $a \parallel b$: concatenation of two bit strings a and b.
- $\mathbf{Z}_n = \{0, 1, \cdots, n-1\}$ and $\mathbf{Z}_n^* = \{x | 1 \le x \le n-1 \ \& \ \gcd(x, n) = 1\}$.
- $|A|$ denotes the bit-length of A for an integer A.
- $k \in_r S$ denote that k is chosen at random over the set S.

The rest of this paper is organized as follows. We describe KCDSA parameters and the detailed algorithm of KCDSA in Section 2 and its elliptic curve variant (EC-KCDSA) in Section 3. The efficiency and security aspects of KCDSA are discussed in Sections 4 and 5, respectively. Finally, we conclude in Section 6.

2 KCDSA

2.1 KCDSA Parameters

KCDSA parameters can be divided into domain parameters and user parameters. By domain we mean a group of users who shares the same public parameters (domain parameters). Domain may consist of a single user if the user uses its own public parameters. User parameters denote parameters which are specific to each user and cannot be shared with others. These parameters must be established before normal uses of digital signatures by some trusted authorities and/or by users. KCDSA makes use of the following domain and user parameters (see Appendix A for a procedure that can be used to generate domain parameters):

Domain Parameters: p, q, g such that

- p : a large prime such that $|p| = 512 + 256i$ for $i = 0, 1, \cdots, 6$. That is, the bit-length of p can vary from 512 bits to 2048 bits with increment by a multiple of 256 bits.
- q : a prime factor of $p - 1$ such that $|q| = 128 + 32j$ for $j = 0, 1, \cdots, 4$. That is, the bit-length of q can vary from 128 bits to 256 bits with increment by a multiple of 32 bits. Further, it is required that $(p-1)/2q$ should be a prime or at least all of its prime factors should be greater than q.[1]
- g : a base element of order q in $GF(p)$, i.e., $g \neq 1$ and $g^q = 1 \bmod p$.

User Parameters: x, y, z such that

- x : signer's private signature key such that $x \in_r \mathbf{Z}_q^*$.
- y : signer's public verification key computed by $y = g^{\overline{x}} \bmod p$, where $\overline{x} = x^{-1} \bmod q$.[2]

[1] This restriction on the size of prime factors of $(p-1)/2q$ is to take precautions against possible attacks using small order subgroups of $\mathbf{Z}_p^*$ in various applications of KCDSA (see [9] for details).

[2] Notice that there is essentially no difference in the signature algorithm if the secret-public key pair $\{x, y = g^{\overline{x}} \bmod p\}$ is represented by $\{\overline{x}, y = g^x \bmod p\}$. We simply adopted the above notation to clarify (to the public unaware of cryptography) that we only need x for a signing purpose. This kind of key pair may be undesirable if the same key is to be used for other purposes as well (e.g., key exchange or entity authentication). However, it is a common practice in cryptographic protocol designs that the same key should not be used for different purposes.

- z : a hash-value of $Cert\_Data$, i.e., $z = h(Cert\_Data)$. Here $Cert\_Data$ denotes the signer's certification data, which should contain at least signer's distinguished identifier, public key y and the domain parameters $\{p, q, g\}$.

KCDSA is a signature algorithm in which the public key is validated by means of a certificate issued by some trusted authority. The X.509-based certificate may be used for this purpose. In this case, the $Cert\_Data$ can be simply the formatted certification data defined by X.509.

KCDSA also requires a collision-resistant hash function which produces $|q|$-bit outputs. Since q can vary in size from 128 bits to 256 bits with increment by a multiple of 32 bits, we need a family of hash functions or a hash function which can produce variable length outputs up to 256 bits.

2.2 The Signature Algorithm

Signature Generation: The signer can generate a signature $\{r\|s\}$ for a message m as follows:

1. randomly picks an integer k in $\mathbf{Z}_q^*$ and computes $w = g^k \bmod p$,
2. computes the first part r of the signature as $r = h(w)$,
3. computes $e = r \oplus h(z\|m) \bmod q$,
4. computes the second part s of the signature as $s = x(k - e) \bmod q$, and
5. if s=0, then repeats the above process.[3]

The computation of w is the most time-consuming operation in the signing process. However, since the first two steps can be performed independent of a specific message to be signed, we may precompute and securely store the pair $\{r, k\}$ for fast on-line signature generation. The above signing process can be described in brief by the following two equations:

$$\begin{aligned} r &= h(g^k \bmod p) \text{ with } k \in_r \mathbf{Z}_q^*, \\ s &= x(k - r \oplus h(z\|m)) \bmod q. \end{aligned}$$

Signature Verification: On receiving $\{m\|r\|s\}$, the verifier can check the validity of the signature as follows:

1. first checks the validity of the signer's certificate, extracts the signer's certification data $Cert\_Data$ from the certificate and computes the hash value $z = h(Cert\_Data)$.[4]
2. checks the size of r and s : $0 \le r < 2^{|q|}$, $0 < s < q$,
3. computes $e = r \oplus h(z\|m) \bmod q$,
4. computes $w' = y^s g^e \bmod p$, and

[3] Even in the case of $s = 0$, there is no possibility of the secret x being disclosed. However, the signature with $s = 0$ should not be accepted as valid, since we cannot identify the signer in this case.

[4] Note that a certificate corresponds to a trusted authority's signature for the formatted data containing all information required to bind the public key and related parameters/attributes to the key owner's identity. Therefore, the computation of z can be in fact part of the certificate validation process by taking $Cert\_Data$ as the formatted data to be signed.

5. finally checks if $r = h(w')$.

The pair $\{r \| s\}$ is a valid signature for m only if all the checks succeed. The above verifying process can be described in brief by the following equations:

$$
\begin{aligned}
e &= r \oplus h(z \| m), \\
r &= h(y^s g^e \bmod p) \ ?
\end{aligned}
$$

For comparison, we summarized three signature standards, DSA, GOST and KCDSA, in Table 1.

Schemes	$\lvert p \rvert$	$\lvert q \rvert$
DSA	$512 + 64i \ (i = 0, 1, \cdots, 8)$	160
GOST	512 or 1024	256
KCDSA	$512 + 256i \ (i = 0, 1, \cdots, 6)$	$128 + 32i \ (i = 0, 1, \cdots, 4)$

Schemes	Signature Generation	Signature Verification
	private key : $x \in_r \mathbf{Z}_q^*$	public key : $y = g^x \bmod p$
DSA [20]	$k \in_r \mathbf{Z}_q^*$ $r = (g^k \bmod p) \bmod q$ $s = k^{-1}(rx + h(m)) \bmod q$	$(y^{s^{-1}r} g^{s^{-1}h(m)} \bmod p) \bmod q = r$?
GOST [11]	$k \in_r \mathbf{Z}_q^*$ $r = (g^k \bmod p) \bmod q$ $s = rx + kh(m) \bmod q$	$(y^{-rh(m)^{-1}} g^{sh(m)^{-1}} \bmod p) \bmod q = r$?
KCDSA	private key : $x \in_r \mathbf{Z}_q^*$	public key : $y = g^{x^{-1}} \bmod p$
	$k \in_r \mathbf{Z}_q^*$ $r = h(g^k \bmod p)$ $s = x(k - r \oplus h(z \| m)) \bmod q$	$h(y^s g^{r \oplus h(z \| m)} \bmod p) = r$?

Table 1: Comparison of DSA, GOST and KCDSA

3 Elliptic Curve KCDSA(EC-KCDSA)

Much attention has been paid to elliptic curve cryptosystems in recent years, due to their stronger security and higher speed with smaller key size. An elliptic curve variant of KCDSA (EC-KCDSA for short) was not considered during the standardization process. However, we have recently worked on an alternative implementation of KCDSA over elliptic curves and completed a high-level specification of EC-KCDSA. This elliptic curve variant is described below.

3.1 EC-KCDSA Parameters

The domain parameters of EC-KCDSA mainly consist of parameters to define a finite field and ones to define an elliptic curve over the finite field. The following parameters need to be defined for elliptic curve KCDSA.

- a prime p and a positive integer m $(m \geq 1)$ defining a finite field $GF(p^m)$.

- a monic irreducible polynomial $f(x)$ of degree m over $GF(p)$ if $m > 1$.

- coefficients $a, b (\in GF(p^m))$ defining an elliptic curve E over $GF(p^m)$, $E(GF(p^m))$

$$E = E_{a,b} : \begin{cases} Y^2 + XY = X^3 + aX^2 + b & (b \neq 0) & \text{if } p{=}2 \\ Y^2 = X^3 + aX + b & (4a^3 + 27b^2 \neq 0 \text{ in } GF(p^m)) & \text{if } p > 3. \end{cases}$$

- a prime q dividing $\#E(GF(p^m))$, where $\#E(GF(p^m))$ denotes the order of the elliptic curve E (the total number of points on $E(GF(p^m))$).

- a point $G = (g_x, g_y)$ in E generating a cyclic subgroup of prime order q.[5]

In an implementation for a general purpose use, we recommend to choose q as a prime of at least 160 bit length.

Once domain parameters are determined, each user can generate its own public and private parameters to join the system. The user parameters of EC-KCDSA consist of the following:

- a private signature key x chosen at random over $\mathbf{Z}_q^*$.

- the public verification key Y computed by $Y = \overline{x}G$ in E, where $\overline{x} = x^{-1} \bmod q$.

- the hashed certification data z.

3.2 The Signature Algorithm

The signing and verifying processes in EC-KCDSA are almost the same as those of KCDSA, except for the change of group operations. That is, the underlying group is changed from the multiplicative group of a prime field into the additive group of elliptic curve points.

Signature Generation: To generate a signature $\{r\|s\}$ on message m, the signer performs the following:

1. randomly picks an integer k in $\mathbf{Z}_q^*$ and computes $W = kG = (w_x, w_y)$ in E,

2. computes the first part r of the signature as $r = h(W) = h(w_x\|w_y)$,

3. computes $e = r \oplus h(z\|m) \bmod q$,

4. computes the second part s of the signature as $s = x(k - e) \bmod q$, and

5. if s=0, then repeats the above process.

As in KCDSA, the first two steps may be performed off-line for faster real-time signature generation. The signing process can be described in brief by the following two equations:

$$\begin{aligned} r &= h(kG) \text{ with } k \in_r \mathbf{Z}_q^*, \\ s &= x(k - r \oplus h(z\|m)) \bmod q. \end{aligned}$$

[5]Suppose that $\#E(GF(p^m)) = kq$ for some integer k. According to [9], the effective key length may be reduced from $|q|$ to $|q| - |k|$ bits in some applications of signature schemes. Therefore, considering wide applications of signature schemes, we strongly recommend that an elliptic curve should be chosen so that the size of k is as small as possible. Ideally, $|q| = |\#E(GF(p^m))|$.

Note that we do not require conversion of elliptic curve points to integers or vice versa. The coordinates w_x, w_y of the point W will be represented as p-ary strings in $GF(p^m)$. However, we do not care about their representations. We simply concatenate the two coordinates as they are and hash the resulting string to obtain r. This is another advantage of EC-KCDSA over EC-DSA[21].

Signature Verification: To verify the signature $\{m\|r\|s\}$, the verifier performs the following:

1. first checks the validity of the signer's certificate, extracts the signer's certification data $Cert\_Data$ from the certificate and computes the hash value $z = h(Cert\_Data)$.
2. checks the size of r and s : $0 \le r < 2^{|q|}$, $0 < s < q$,
3. computes $e = r \oplus h(z\|m) \bmod q$,
4. computes $W' = sY + eG$ in E, and
5. finally checks if $r = h(W')$.

The above verifying process can be described in brief by

$$\begin{aligned} e &= r \oplus h(z\|m) \bmod q, \\ r &= h(sY + eG)\ ? \end{aligned}$$

For comparison, we summarized two signature algorithms, EC-DSA[21] and EC-KCDSA, in Table 2.

Schemes	Finite fields	q
EC-DSA	$GF(p)$ or $GF(2^m)$	$q > 2^{160}$
EC-KCDSA	$GF(p)$, $GF(2^m)$ or $GF(p^m)$	$\|q\| = 128 + 32i\ (i = 0, 1, \cdots, 4)$

Schemes	Signature Generation	Signature Verification
EC-DSA[21]	private key : $x \in_r \mathbf{Z}_q^*$	public key : $Y = xG$
	$k \in_r \mathbf{Z}_q^*$ $r = \pi(kG) \bmod q$ $s = k^{-1}(rx + h(m)) \bmod q$	$u_1 = s^{-1}r \bmod q$ $u_2 = s^{-1}h(m) \bmod q$ $\pi(u_1Y + u_2G) \bmod q = r$?
EC-KCDSA	private key : $x \in_r \mathbf{Z}_q^*$	public key : $Y = \overline{x}G$ ($\overline{x} = x^{-1} \bmod q$)
	$k \in_r \mathbf{Z}_q^*$ $r = h(kG)$ $s = x(k - r \oplus h(z\|m)) \bmod q$	$e = r \oplus h(z\|m) \bmod q$ $h(sY + eG) = r$?

Table 2: Comparison of EC-DSA and EC-KCDSA

Note that $\pi()$ is a function which converts an elliptic curve point to an integer. In case of an elliptic curve defined over $GF(p)$, $\pi((x_1, y_1)) = x_1$. In case of $GF(2^m)$, since x_1 can be represented as a binary string $(s_{m-1}s_{m-2}\cdots s_1s_0)$, $\pi((x_1, y_1)) = \sum_{i=0}^{m-1} s_i 2^i$.

4 Efficiency Considerations

KCDSA is designed to avoid the evaluation of multiplicative inverse in normal uses. It is only needed at the time of key pair generation. For comparison, in DSA a multiplicative inverse mod q needs to be evaluated each time a signature is generated or verified and in GOST each time a signature is verified (see Table 1 and Table 2). Evaluating an inverse mod q would take very little portion in the overall workload of signing/verifying on most general purpose computers. However, it may be quite expensive in a limited computing environment such as smart cards (see [16] for various comments on DSS including debates on the use of inverse). On the other hand, KCDSA needs one more call for a hash function to digest a message of length $|p|$ during both the signature generation and the verification process. However, this will not cost much in any environment.

We have implemented various signature schemes in the C language with inline assembly [10] and measured their timings on 90 MHz Pentium and 200 MHz Pentium Pro. The result is shown in Table 3.[6] As can be expected, KCDSA and DSA show almost the same performance figures, but GOST runs about 63 % ($\approx \frac{160}{256}$) slower than KCDSA/DSA since it uses a 256-bit prime q. For comparison, we also measured the speed of RSA for the same size of modulus. Note that signature generation can be substantially speeded up in both RSA and ElGamal-type schemes: We can use the Chinese Remainder Theorem to speed up RSA signature generation and the precomputation technique [8] to speed up signature generation in ElGamal-type schemes. These performance figures are also shown after '/' in Sign columns. The table shows that KCDSA/DSA can sign about 6 to 10 times faster than RSA, while RSA can verify about 12 to 13 times faster than KCDSA/DSA (RSA verification key: $e = 2^{16} + 1$).

Compared to usual DL-based signature schemes, we need in general one more computational step in the elliptic curve variants, i.e., the step of converting an elliptic curve point to integer during the computation/reconstruction of the first part r of the signature. This conversion step may degrade the performance more or less, depending on the underlying finite field. For example, it is trivial to convert a point on $E(GF(2^m))$ to integer. However, in the case of $E(GF(p^m))$ for $p > 3$ the conversion step may take nontrivial processing time. One advantage of EC-KCDSA is that there is no need of elliptic curve point to integer conversion. This is thanks to the use of a hash function for the computation of r. In EC-KCDSA, the two coordinates w_x and w_y, represented as p-ary strings, are simply concatenated string by string and the resulting string is hashed to produce the first part r of the signature.

5 Security Considerations

5.1 Security Proof under Random Oracle Model

Recently two variants of ElGamal-like signature schemes have been proven secure against adaptive attacks for existential forgery under the random oracle model [3], where the hash function is replaced with an oracle producing a random value for each new query. In the first variant, $h(m)$ is replaced with $h(m\|r)$ as in the Schnorr signature scheme. This variant was proven secure by Pointcheval and Stern [14] at Eurocrypt'96. The other variant is due to Brickell [4] at Crypto'96, where he claimed that the variant of DSA with $r = (g^k$ mod

[6] We used SHA-1 for hashing with a very short message in all the signature schemes. Multiplicative inverses were computed using an extended Euclidean algorithm.

		Pentium/90		Pentium Pro/200	
Algorithm	Lang.	Sign	Verify	Sign	Verify
DSA	C	289 / 57.8°	359	95.0 / 18.9	117
($\|q\| = 160$)	D	148 /29.8	182	47.3 /9.7	58.0
	A	64.0 /13.7	79.1	17.5 /3.9	21.7
GOST	C	457 / 87.8	559	147 / 28.0	181
($\|q\| = 256$)	D	236 /44.3	287	73.4 /14.0	92.3
	A	105 /19.1	125	27.2 /5.2	35.3
KCDSA	C	287 / 56.2	359	93.3 / 18.0	116
($\|q\| = 160$)	D	145 /28.0	185	46.4 /9.0	57.4
	A	62.8 /12.4	77.7	17.0 /3.3	20.9
RSA	C	1730 / 502*	25.8	568 / 163	8.6
($e = 2^{16} + 1$)	D	878 / 254	15.8	279 / 83.5	5.3
	A	378 / 114	6.0	103 / 33.1	1.7

Notes :
C = C only,
D = C with double digit option (__int64) provided by MSVC,
A = C with partial inline assembly.
* used CRT for signature generation.
∘ used a precomputated table of 32 KBytes (6×4 config., see [8]).

Table 3: Speed of various DL and IF signature schemes for 1024-bit moduli (in msec)

p) mod q replaced by $r = h(g^k \bmod p)$ is also secure in the random oracle model (see [15] for its proof by Pointcheval and Vaudenay). We followed the latter approach to ensure the security of the overall design of KCDSA. From the proof under the random oracle model we can be assured that KCDSA will be secure provided that the hash function used has no weakness.

5.2 Security against Parameter Manipulation

There have been published a lot of weaknesses in the design of discrete log-based schemes due to the use of unsafe parameters (later shown insecure) (e.g., see [13, 2, 1, 19, 9]). Note that generating public parameters at random so that they do not have any specific structure is very important for security, even with a provably secure scheme (compare the results from [2] and [14]. see also [18]). KCDSA is designed to be secure against all these potential weaknesses. The (proposed) standard recommend to use the strongest form of primes [9], i.e., primes p, q such that $(p-1)/2q$ is also a prime or at least its prime factors are all greater than q. It also specifies a procedure that can be used for generation of such primes (see Appendix A). The certificate produced by this procedure can be used to verify proper generation of the parameters. Considering current algorithms and technology for finding discrete logarithms (see [12]), we recommend to use a modulus p of size 1024 bits and an auxiliary prime q of 160 bits for moderate security in most applications.

The use of the parameter $z = h(Cert\_Data)$ as a prefix message for hashing provides several advantages without much increase of computational/operational overheads.[7]

[7] In the present standard the hashed certification data z is used as part of message (i.e., $z||m$ is treated

It effectively prevents possible manipulations during parameter generation, such as hidden collisions in DSS [19], since $Cert\_Data$ contains p, q, g and y. In addition, the use of z restricts the collision search in the hash function to a specific signer, since each signer uses his/her own prefix z to produce a hash code for his/her message. To see its usefulness, suppose that in the case of using the usual hash code $h(m)$ a collision is found for a specific pair of messages. Also suppose that one message out of the pair is a comfortable message that anyone can sign without reluctance. Then the collision can be used to any user to claim that the signature is for the harmful message. Realization of this scenario may be catastrophic, for example, if there exists some powerful organization willing to invest a huge amount of money to find collisions (the organization might find some unpublished weakness in the hash function which can substantially reduce the time for exhaustive search). Our new hash mode with an user-specific prefix can effectively thwart such a trial of total forgery unless a serious weakness is found for the hash function.

5.3 Security of EC-KCDSA

EC-KCDSA also preserves most security aspects of KCDSA. In other words, it is provably secure under some ideal assumption on the hash function used, and the use of the parameter $z = h(Cert\_Data)$ effectively prevents any potential attack using parameter manipulation. As noted before, it would be safer to choose an elliptic curve E and a point G so that $|\#(E)/q|$ is as small as possible.

In general, the security of EC-KCDSA with well-chosen parameters (see [22], [23], [24] and [25]) will be stronger than that of KCDSA if both use the same size of q.

6 Conclusion

We described the proposed digital signature standard for Korean community (KCDSA) and its elliptic curve variant (EC-KCDSA), and discussed their security and efficiency aspects. The KCDSA algorithm is now close to publication as one of Korean Information and Communication Standards (KICS) and the EC-KCDSA will also be one of them in the near future. The signature algorithms are expected to be widely used by commercial and government sectors in Korea. No patent related with KCDSA/EC-KCDSA has been submitted. We thus hope this contribution to IEEE P1363a can promote practical applications of KCDSA/EC-KCDSA in other countries as well.

Acknowledgement

We are grateful to Yuliang Zheng, Claus-Peter Schnorr, Serge Vaudenay, Ross Anderson, Peter Landrock, Thomas A. Berson, Moti Yung, Kaisa Nyberg, Mark King and many others for their helpful comments during the development of KCDSA.

as a message to be signed). However, it may be more desirable to separate z from the message to be signed. For example, we may use z itself as an user-specific IV or complete z into one block by zero-padding and use $h(z||pad)$ as an user-specific IV. These variants will be further discussed in the next revision.

References

[1] R.Anderson and S.Vaudenay, Minding your p's and q's, In *Advances in Cryptology - ASIACRYPT'96*, LNCS 1163, Springer-Verlag, pp.15-25 (1996).

[2] D.Bleichenbacher, Generating ElGamal signatures without knowing the secret, In *Advances in Cryptology - EUROCRYPT'96*, LNCS 1070, Springer-Verlag, pp.10-18 (1996).

[3] M.Bellare and P.Rogaway, Random oracles are practical: a paradigm for designing efficient protocols, In *Proc. of 1st ACM Conference on Computer and Communications Security*, pp.62-73 (1993).

[4] E.F.Brickell, Invited lecture given at Crypto'96, unpublished.

[5] W.Diffie and M.Hellman, New directions in public key cryptography, *IEEE Trans. on Information Theory*, IT-22 (6), pp.644-654 (1976).

[6] T.ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Inform. Theory*, IT-31, pp.469-472 (1985).

[7] Knuth, *The Art of Computer Programming*, Vol. 2, Addison-Wesley (1981).

[8] C.H.Lim and P.J.Lee, More flexible exponentiation with precomputation, In *Advances in Cryptology - CRYPTO'94*, LNCS 839, Springer-Verlag, pp.95-107 (1994).

[9] C.H.Lim and P.J.Lee, A key recovery attack on discrete log based schemes using a prime order subgroup, In *Advances in Cryptology - CRYPTO'97*, LNCS 1294, Springer-Verlag, pp.249-263 (1997).

[10] C.H.Lim, H.S.Hwang and P.J.Lee, Fast modular reduction with precomputation, *Proc. of 1997 Korea-Japan Joint Workshop on Information Security and Cryptology (JW-ISC'97)*, Seoul, pp.65-79 (1997).

[11] M.Michels, D.Naccache and H.Petersen, GOST 34.10 - A brief overview of Russia's DSA, *Computers and Security*, 15(8), pp.725-732 (1996).

[12] A.M.Odlyzko, The future of integer factorization, *CryptoBytes*, 1(2), pp.5-12 (1995).

[13] P.C.van Oorschot and M.J.Wiener, On Diffie-Hellman key agreement with short exponents, In *Advances in Cryptology - EUROCRYPT'96*, LNCS 1070, Springer-Verlag, pp.332-343 (1996).

[14] D.Pointcheval and J.Stern, Security proofs for signature schemes, In *Advances in Cryptology - EUROCRYPT'96*, LNCS 1070, Springer-Verlag, pp.387-398 (1996).

[15] D.Pointcheval and S.Vaudenay, On provable security for digital signature algorithms, a manuscript, available from *http://www.dmi.ens.fr/~pointche/* (1996).

[16] R.L.Rivest / M.E.Hellman / J.C.Anderson, Responses to NIST's proposal, *Comm. ACM*, 35(7), pp.41-52 (1992).

[17] R.L.Rivest, A.Shamir and L.Adleman, A method for obtaining digital signatures and public key cryptosystems, *Commun. ACM*, 21(2), pp.120-126 (1978).

[18] J.Stern, The validation of cryptographic algorithms, In *Advances in Cryptology - ASIACRYPT'96*, LNCS 1163, Springer-Verlag, pp.301-310 (1996).

[19] S.Vaudenay, Hidden collisions on DSS, In *Advances in Cryptology - CRYPTO'96*, LNCS 1109, Springer-Verlag, pp.83-88 (1996).

[20] NIST, Digital signature standard, *FIPS PUB 186* (1994).

[21] ANSI, X9.62-1998, Public Key Cryptography For The Financial Services Industry : The Elliptic Curve Digital Signature Algorithm (ECDSA), April 14, 1998.

[22] A. Menezes, T. Okamoto, and S. A. Vanstone, Reducing elliptic curve logarithms to logarithms in a finite field, *Proceedings of the 23rd ACM Symp. Theory of Computing*, 1991.

[23] Semaev, I. A., Evaluation of discrete logarithms ina group of p-torsion points of an elliptic curves in characteristic p, *Math. Comp.*, 67(1998), pp.353-356.

[24] Smart, N. P., The discrete logarithm problem on elliptic curves of trace one, 1997, *preprint*.

[25] T. Satoh and K. Araki, Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves, 1998, *preprint*.

A Domain Parameter Generation for KCDSA

During the KCDSA initialization stage, a trusted authority in each domain has to generate and publish p, q, g such that

- p is a prime of specified length such that a prime q of specified length divides $p-1$ and that all prime factors of $(p-1)/2q$ are greater than q.
- g is a generator of a subgroup of $\mathbf{Z}_p^*$ of order q, i.e., g is an element of $\mathbf{Z}_p$ such that $g^q = 1 \bmod p$ and $g \neq 1$. Such a g can be generated by testing $g^{(p-1)/q} = 1 \bmod q$ with random $1 < g < p$.

As an example, we describe a method for generating primes p, q such that $(p-1)/2q$ is also prime. Let $PRG(s,n)$ denote a pseudorandom number generator on input s generating an n-bit random number, defined by:

$$\begin{aligned} v_i &= h(s + i \bmod q) \text{ for } i = 0, 1, \cdots, k-1, \\ v_k &= h(s + k \bmod q) \bmod 2^r, \\ PRG(s,n) &= v_k \parallel v_{k-1} \parallel \cdots v_1 \parallel v_0, \end{aligned}$$

where $k = \frac{n}{|q|}$ and $r = n \bmod |q|$.

A.1 The Algorithm

The procedure for generating p, q (of size $|p|, |q|$, respectively) and g is as follows (see also Figure 1):

1. choose an arbitrary integer s of at least $|q|$ bits.

2. initialize five counters: $tCount = rCount = 1$, $pCount = qCount = gCount = 0$.

3. form $Seed$ for PRG as:

$$\begin{aligned} w_2 &= 0x00 \parallel i \parallel j \parallel tCount, \\ w_1 &= rCount \parallel pCount, \\ w_0 &= qCount \parallel gCount \parallel 0x00, \\ Seed &= s \parallel w_2 \parallel w_1 \parallel w_0, \end{aligned}$$

   where i and j are 8 bit numbers such that $|p| = 512+256i$ and $|q| = 128+32j$, $tCount$ and $gCount$ are 8 bits long, and $pCount$, $qCount$ and $rCount$ are 16 bits long. It is assumed that $Seed$ is automatically updated whenever any counter is changed.

4. generate a random number r of length $|p| - |q| - 1$ bits as follows:

$$\begin{aligned} u &= PRG(Seed,\ |p| - |q| - 1), \\ r &= 2^{|p|-|q|-2} \vee u \vee 1, \end{aligned}$$

   where $\vee$ denotes bitwise-or.

5. test r for primality (e.g., using the Miller-Rabin probabilistic primality test [7, page 379]). If r is prime, go to step 8.

6. increment $rCount$ by 1.

7. If $rCount < 2048$, go to step 4. Otherwise, go to step 1.

8. set $pCount = 1$ and $qCount = 1$.

9. generate a random number q of length $|q|$ bits using the updated $Seed$ as follows:

$$\begin{aligned} u &= PRG(Seed,\ |q|), \\ q &= 2^{|q|-1} \vee u \vee 1. \end{aligned}$$

10. compute $p = 2qr + 1$. If $|p| < |p|$, go to step 12.

11. test q for primality. If q is prime, go to step 14.

12. increment $qCount$ by 1.

13. If $qCount < 1024$, go to step 9. Otherwise, go to step 15.

14. test p for primality. If p is prime, go to step 19.

15. increment $pCount$ by 1 and set $qCount = 1$.

16. If $pCount < 4096$, go to step 9.

17. increment $tCount$ by 1.

18. If $tCount < 256$, go to step 3. Otherwise, go to step 1.

19. set $gCount = 1$.

20. generate a random number u of length $|p|$ bits using the updated $Seed$ as follows:

$$u = PRG(Seed,\ |p|).$$

21. compute $g = u^{(p-1)/q} \bmod p$. If $g \neq 1$, go to step 24.

22. increment $gCount$ by 1.

23. If $gCount < 256$, go to step 20. Otherwise, go to step 17.[8]

24. terminate with output p, q, g and $Seed$.

The $Seed$ output can serve as a certificate for proper generation of the parameters p, q and g. Anyone can check that p, q and g are generated as specified, since $Seed$ contains all necessary information to verify their proper generation.

A.2 Numerical Example

For example, the following parameters ($|p| = 1024, |q| = 160$) were generated using the described algorithm, where we the initial user input s was taken as the first 160 bits of the fractional part of $\pi = 3.14159\cdots$. ¿From the seed, we can see that $r = (p-1)/2q$ was found by testing 991 random numbers ($rCount = 0x3df = 991$) and p was found by testing 1192 primes of q ($pCount = 0x77c = 1192$) and so on. It is easy to verify that these parameters are generated according to the above procedure.

```
Seed = 243f6a88 85a308D3 13198a2e 03707344 a4093822
       00020101 03df077c 00d10100
   p = a2951279 6e6cf682 fd9e3348 24859dfd 93299a22 7d9d6c97 226B9595
       1725c3B5 3098ceaa 3e6a0241 d0c30586 61769311 9db2e9bc 2f9cad43
       9f17fe3B 8a54f711 820421a0 394218e8 3186641d 00373299 08ab8D2f
       97ffb1c7 5afaaba3 5e356ae8 7f83d2f8 d79d031c d814318f e7865810
       16a3c871 a159056c 70722a62 cb89694f

   q = ada5ff8f 174cab84 0c846634 dede6e81 5ac8f6ef

   g = 1b2f2d3b a6551ffd a74ca533 011f1a92 8277d572 67297496 78a42bda
       5ba6c181 9cf283ee 14a3fb44 dacbe42b b9720d2d 7137c81e 69cfc7cf
       20a41bb1 e117fa7d 9b8d0cb0 73a91e51 15c08db8 60be3633 67a08ac2
       b59137c2 0ccf54b9 0dbc2c8c 90958555 d76c0020 2798282a 23cafc54
       7c7e7820 cf979902 2d3cde88 52d13753
```

[8] The probability of $gCount$ exceeding 255 is negligible ($gCount = 1$ for almost all cases). For completeness, we simply make the control to go back to step 3 in such an exceptional case (through steps 17, 18, 3).

Conditions:

|p| = 512 + 256i (i=0, 1, ..., 6)

|q| = 128 + 32j (j=0, 1, ..., 4)

r = (p-1)/2q : prime

Seed = s (user input) || 0x00 || i || j || tCount || rCount || pCount || qCount || gCount || 0x00

(tCount, gCount : 8 bits, pCount, qCount, rCount : 16 bits)

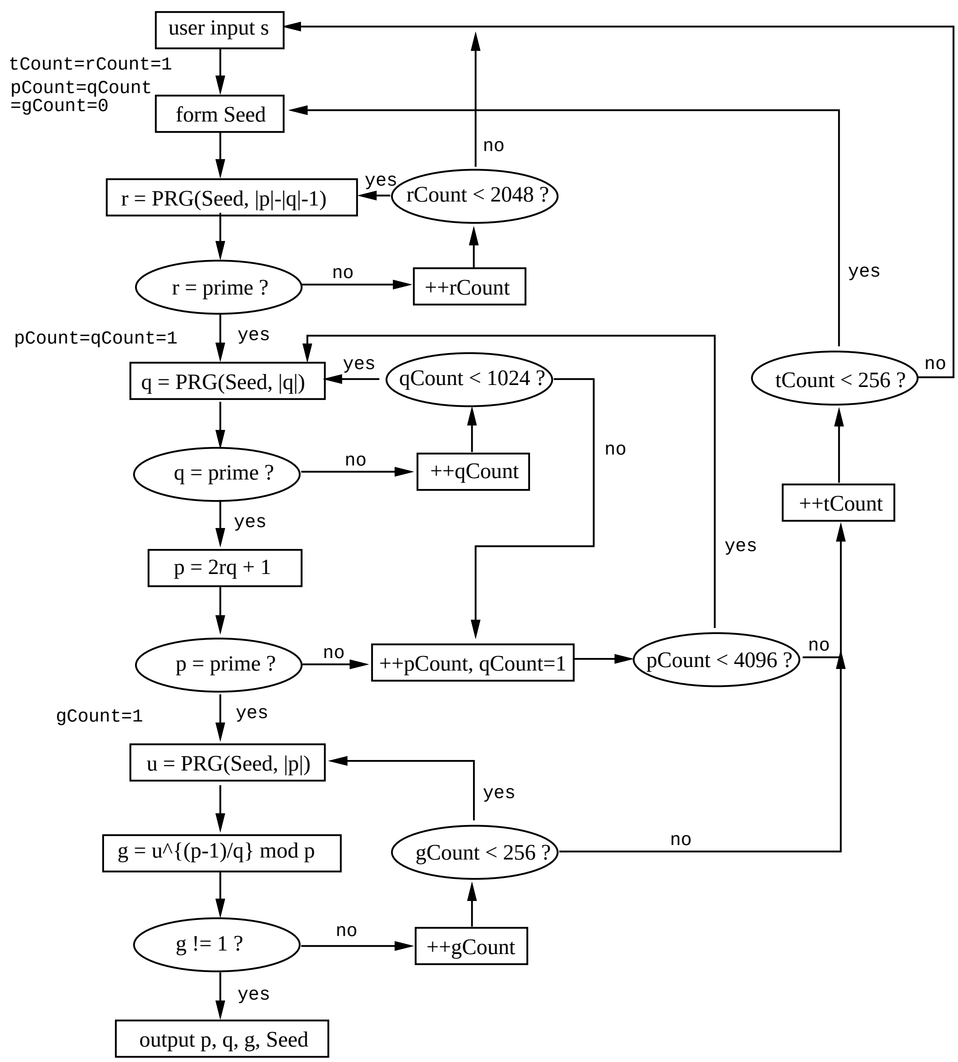

Figure 1: Flow chart for generation of p, q, g