# A Security Proof of KCDSA using an extended Random Oracle Model

Vikram Singh *

**Abstract**

We describe a tight security reduction to the discrete logarithm problem for KCDSA under an extended Random Oracle Model. This is achieved by generalising the signature scheme and producing a security proof for the generalised scheme. We require the application of Randomized Hashing. We also introduce a Challenger to the Random Oracle Model, who is external to the Simulator and Adversary. The Challenger provides oracle returns for one hash function, and challenges which have a low probability of being met. On presentation of a forged signature the Simulator either identifies an edge case which allows solving of a challenge, or solves the discrete logarithm problem. Hence the tight reduction.

*Keywords: Cryptography, Provable Security, KCDSA, Tight Reduction, Discrete Logarithm Problem, Random Oracle Model*

* vs77814@gmail.com. Completed in 2008 based on studies at RWTH Aachen University, 2005-2007.

# Contents

# 1 Introduction

This paper describes a tight security reduction to the discrete logarithm problem for a generalised KCDSA scheme (for which KCDSA and EC-KCDSA are specific cases). The result may alternatively be interpreted as providing a tight security proof for KCDSA under an extended Random Oracle Model assumption. This result is not intended to promote a new signature scheme but to add to the perceived security of KCDSA. For details of the KCDSA algorithm see [1]. It may be assumed throughout that KCDSA is considered with the method of Randomized Hashing applied.

This paper is divided into 4 sections. Section 2 describes the generalisation of KCDSA. Section 3 gives the tight reduction to the discrete logarithm problem and Section 4 gives the conclusion.

# 2 The Generalised KCDSA Signature Scheme

## 2.1 Setup

The algorithm is performed in a cyclic group $\mathbb{G}$ of order $q$, generated by $g$. Let elements of $\mathbb{G}$ have bit length $l_p$ and q be a $l_q$-bit prime. The group is viewed multiplicatively. Let $\mathbb{L}$ be the space of $l_q$-bit values and let the space of possible messages be $\mathbb{M}$.

Three hash functions are required:

$$\begin{aligned} H_1 &: \mathbb{Z}_q \rightarrow \mathbb{L} \\ H_2 &: \mathbb{Z}_q \times \mathbb{Z}_q^* \rightarrow \mathbb{L} \\ H_3 &: \mathbb{M} \rightarrow \mathbb{L} \end{aligned}$$

The hash functions $H_1$ and $H_2$ are related hash functions. The following relationship holds, where $y$ is the public key defined in Section 2.2 below:

$$g^k = g^e y^s \Rightarrow H_1(k) = H_2(e, s) \tag{1}$$

A further extension function $G$ is also required for applying Randomized Hashing:

$$G : \mathbb{L} \rightarrow \mathbb{M}$$

For more details on this extension and Randomized Hashing see [2].

## 2.2 Key Generation

The private key is a random number $x \in \mathbb{Z}_q^*$. The public key is $y = g^{\frac{1}{x}}$.

### 2.3 Signature Generation

To sign a message $m \in \mathbb{M}$, the signer generates $k \in \mathbb{Z}_q$ randomly. The signer computes $r = H_1(k)$ and extends $r$ to $rv = G(r) \in \mathbb{M}$. The signer computes:

$$\begin{aligned} e &= r \oplus H_3(rv \oplus m) \mod q \\ s &= x(k - e) \in \mathbb{Z}_q \end{aligned}$$

If $s = 0$, repeat the process. The signature of $m$ is $(r, s)$.

### 2.4 Signature Verification

To verify the signature, $(m, r, s)$, the verifier first checks that $s \in \mathbb{Z}_q^*$, $r \in \mathbb{L}$ and $m \in \mathbb{M}$. The Simulator then computes

$$e = r \oplus H_3(G(r) \oplus m) \mod q$$

The signature is valid if $H_2(e, s) = r$. This relation holds for a valid signature as $g^e y^s = g^k$, thus $H_2(e, s) = H_1(k) = r$

### 2.5 Relationship with KCDSA

We note that if $H_1(k) = h(g^k)$, $H_2(e, s) = h(g^e y^s)$, $H_3(m) = h(z||m)$ where $z$ is some hash of the public certificate data, this algorithm is the Randomized Hashing version of KCDSA (or EC-KCDSA in an alternative group).

## 3 Security Proof of the Generalised KCDSA Algorithm

This section describes a tight reduction of KCDSA to the discrete logarithm problem. In this case, the discrete logarithm problem solved is recovery of the secret $x$.

### 3.1 The Assumptions and Setup

This security proof uses the Random Oracle Model. The hash functions may be chosen randomly from the set of all possible functions, given that the property in Equation 1 holds. Figure 1 provides a pictorial explanation of the setup along with brief descriptions of the values returned by oracle requests.

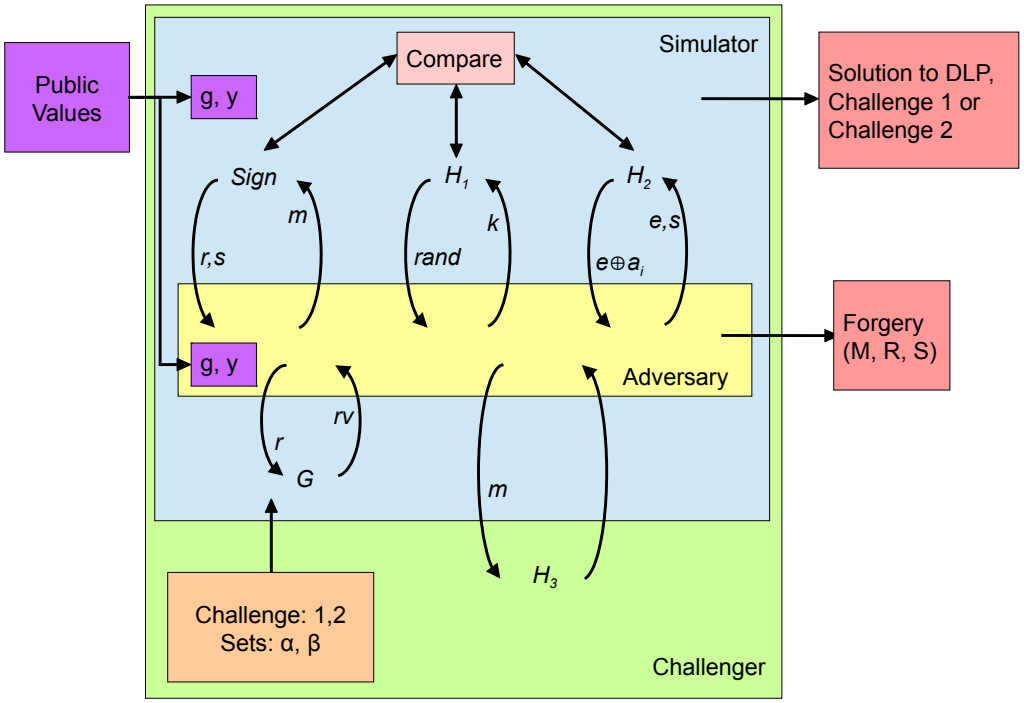


Figure 1: A Security Proof Diagram of Generalised KCDSA

In this security proof there are three components to consider. Firstly, there is an Adversary who is capable of forging a message given (other) message signatures, the public data and the output of the hash functions. The Simulator provides the responses to the Adversary's hash and signature queries. Assume that the Adversary makes $Q_{H_i}$ hash queries to $H_i$, $Q_G$ hash queries to $G$ and $Q_s$ signature queries.

In an extension to the usual setup, there is also a Challenger. The Simulator may choose the output of $H_1$, $H_2$, $G$ and the signature queries. However, the Simulator must forward $H_3$ queries to the 'external' Challenger. The Challenger provides the responses to the Simulator's $H_3$ queries. The purpose of the Challenger in the security proof is to absorb the low probability edge cases wherein an Adversary's forgery does not enable the Simulator to solve the hard problem (in our case, the discrete logarithm problem).

The Challenger is thus named because it issues the following challenges to the Simulator:

*Challenge 1*: Given a random subset of $\mathbb{L}$, $\alpha = \{a_0, \ldots, a_{Q_{H_2}-1}\}$, of size $Q_{H_2}$ find $u \in \mathbb{M}$ such that $H_3(u) = a_i$, for some $i$.

*Challenge 2*: Given a random subset of $\mathbb{M}$, $\beta = \{b_0, \ldots, b_{Q_s-1}\}$, of size $Q_s$ find $v \in \mathbb{M}$ such that $H_3(v) = H_3(b_i)$ and $v \neq b_i$, for some $i$.

As $H_3$ is viewed as a random oracle which is unknown to the Simulator, it may be assumed that solving Challenge 1 or Challenge 2 has probability $\frac{Q_{H_2}+Q_s}{2^{l_q}}$ for each $H_3$ hash query, which can easily be controlled to be a low probability.

## 3.2 The Simulator's Responses

The Simulator's responses to queries from the Adversary are defined in this section. This will allow us to determine which problems the Simulator can solve, given presentation of a forgery. In order to define correct responses a comparison table is needed. The purpose of this table is to ensure that Equation 1 is satisfied in the Simulator's responses to hash and signature queries. An example table is given in Table 1.

| Type | $w$ | $g$ Power | $y$ Power | Return Val |
|---|---|---|---|---|
| $H_1$ | $g^{e_0}$ | $e_0$ | 0 | $t_0$ |
| $H_1$ | $g^{e_1}$ | $e_1$ | 0 | $t_1$ |
| $H_2$ | $g^{e_2}y^{s_2}$ | $e_2$ | $s_2$ | $t_2$ |
| *Sign* | $g^{e_3}y^{s_3}$ | $e_3$ | $s_3$ | $t_3$ |
| $H_1$ | $g^{e_4}$ | $e_4$ | 0 | $t_4$ |
| $H_2$ | $g^{e_5}y^{s_5}$ | $e_5$ | $s_5$ | $t_5$ |

Table 1: A comparison table excerpt for $H_1$ and $H_2$ queries

The $w$ column of the table is consulted with every query to the hash functions $H_1$ and $H_2$; the table holds the inputs and outputs already given for these hash functions. If no entry is found, a new entry is submitted to the table. For a signature query, the '$g$ Power' column is queried and a new entry corresponding to the $H_2$ input and output is added to the table.

The values that are returned with each query are now listed. To begin slet $i = j = 0$:

$H_1(k)$: Simulator calculates $w = g^k$ and consults the comparison table. If $w$ is not present, returns random $t \in \mathbb{Z}_q$ and adds ($H_1$,$w$,$k$,0,$t$) to the table. Otherwise returns corresponding $t$ (to $w$) listed in table.

$H_2(e, s)$: Simulator calculates $w = g^e y^s$ and consults the comparison table. If $w$ is not present, returns $t = e \oplus a_i$, increments $i$, and adds ($H_2$,$w$,$e$,$s$,$t$) to the table. Otherwise returns corresponding $t$ (to $w$) listed in table.

$H_3(m)$: Simulator requests response from Challenger and returns that response.

$G(r)$: Simulator checks if $r$ has been previously submitted. If so, the previously returned value is returned. Else, the Simulator returns a random value $rv \in \mathbb{M}$.

$Sign(m)$: Simulator chooses $r \in \mathbb{L}$ that has not been submitted to the $G$ oracle. Simulator sets $G(r) = b_j \oplus m$, and calculates $e = r \oplus H_3(G(r) \oplus m) \mod q$. Simulator searches for $e$ in the '$g$ Power' column. For each $e_n = e$ found, the Simulator adds the corresponding $s_n$ to a set $\Gamma$. The Simulator then picks an $s$ randomly from the set $\mathbb{Z}_q \backslash \Gamma$. The Simulator increments $j$, adds ($Sign$,$g^e y^s$,$e$,$s$,$r$) to the comparison table, and provides the signature ($r$,$s$) to the Adversary.

To conclude, note that each response will appear random to the Adversary as required by the Random Oracle Model.

## 3.3 The Simulator's Solution

Let us assume that after a time period $\tau$, the Adversary produces a forgery, $(M, R, S)$, with probability $\epsilon$. If the forgery is valid (and $M$ has been previously submitted to the signing Oracle) then we have that:

$$H_2(R \oplus H_3(G(R) \oplus M) \mod q,\ S) = R \tag{2}$$

Set $E = R \oplus H_3(G(R) \oplus M) \mod q$.

Either the Adversary already knows that the output of $H_2$ is $R$, or the Adversary has guessed. The probability that the forger has successfully guessed the output of the hash is around $2^{-l_q}$. Otherwise, $R$ is a 'Return Val' in the comparison table, say $R = t_n$. If $g^{e_n} y^{s_n} \neq g^E y^S$, the return value must also be

found for another table entry or else the Adversary has guessed. So we may assume that $g^{e_n} y^{s_n} = g^E y^S$. The Simulator now considers the corresponding 'Type' of $t_n$.

If 'Type' is $H_1$, then noting that $S \neq 0$ and so $e_n \neq E$ gives:

$$x = \frac{S}{e_n - E}$$

and hence the discrete logarithm problem (DLP) is solved by the Simulator.

If 'Type' is $H_2$, then if $E \neq e_n$:

$$x = \frac{S - s_n}{e_n - E}$$

and hence the DLP is solved by the Simulator. Otherwise, $E = e_n$, $S = s_n$ and by construction, $H_2(E, S) = E \oplus a_i = R$ for some $i$, by virtue of being the forgery, thus $e_n = R \oplus a_i$ for this $i$. Hence $R \oplus H_3(G(R) \oplus M) = R \oplus a_i$ and thus

$$H_3(G(R) \oplus M) = a_i$$

Therefore Challenge 1 is solved.

Finally, if 'Type' is *Sign*, then again if $E \neq e_n$ the DLP is solved by the Simulator. Otherwise $E = e_n$ thus:

$$H_3(G(R) \oplus M) = H_3(G(R) \oplus m)$$

By construction $G(R) = m \oplus b_j$ for some $j$, hence:

$$H_3(b_j \oplus m \oplus M) = H_3(b_j)$$

As $m \neq M$, Challenge 2 is solved.

Thus the Simulator either solves the DLP, Challenge 1 or Challenge 2.

## 3.4 Costing the Simulator

The Simulator only fails if either the $G$ oracle has been exhausted ($Q_G + Q_s > 2^{l_q}$) so a suitable $r$ cannot be found, or that a particular $e$ has been used for all $s$ ($Q_{H_2} > 2^{l_q}$), so $s$ may not be found. These both require sufficiently many queries that we may assume that neither can occur. Hence we can assume that the simulator does not fail. The Simulator either solves the DLP, Challenge 1 or Challenge 2.

The probability that Challenge 1 or Challenge 2 is solved with $Q_{H_3}$ hash queries is approximately $\frac{Q_{H_3}(Q_{H_2}+Q_s)}{2^{l_q}}$. Thus the probability of solving the DLP is:

$$\tilde{\epsilon} \geq \epsilon - \frac{Q_{H_3}(Q_{H_2} + Q_s)}{2^{l_q}} \tag{3}$$

Let $\tau_0$ be the time required to perform group exponentiation in $\mathbb{G}$. So the Simulator requires the following time to complete:

$$\tilde{\tau} = \tau + \tau_0(Q_{H_1} + 2Q_{H_2} + 2Q_s) \tag{4}$$

Thus the security of the signature scheme is tightly reduced to the DLP because the probability of the Simulator solving DLP given a forgery is high, and the time required is little more than required by the forging Adversary.

# 4 Conclusion

This paper has described a tight security reduction for a generalised KCDSA scheme with Randomized Hashing to the discrete logarithm problem. Thus security parameters for the two problems can be tightly related. The result is achieved by adding an external Challenger to the Random Oracle Model, which allows the cases where the Simulator doesn't solve the discrete logarithm problem to be absorbed within low probability challenges.

The purpose of the paper is not to propose a new scheme, but to analyse the security of the existing KCDSA scheme. As such, there are two ways the result could be viewed:

1. The result can be seen as a security proof for KCDSA with the Random Oracle Model assumption. An oracle in the model has been extended to encompass not only the usual hash function but also the input exponentiation. However, this does not invalidate the model.

2. The result could be seen as irrelevant to the KCDSA signature scheme. When related to KCDSA, the model assumes that only exponentiations of $g$ and $y$ may be hashed, and this is clearly not the case in the proof.

We would argue that the second of these is a flawed argument: one could deconstruct any hash function by removing the first few steps of the hash and in doing so, the model under which one is working is ignored. However, to argue this point, one is arguing that the Random Oracle Model is flawed, which may be true, but no more so than when used with any other signature scheme. We would argue that this paper suggests that KCDSA is as 'provably' secure as any other scheme with a security proof in the Random Oracle Model.

# References


[1] Chae Hoon Lim. The revised version of KCDSA. *Unpublished Manuscript available from http://dasan.sejong.ac.kr/~chlim/pub/kcdsa1.ps*, 2000.

[2] Quynh Dang. Randomized Hashing Digital Signatures. *NIST Draft Special Publication 800-106*, 2007.