Standards for Efficient Cryptography

SEC 4: Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV)

Certicom Research

Contact: Matthew Campagna (mcampagna@rim.com)

January 24, 2013
Version 1.0

Contents

List of Figures

1 Introduction

1.1 Overview

This document specifies the Elliptic Curve Qu-Vanstone implicit certificate scheme (ECQV). The ECQV implicit certificate scheme is intended as a general purpose certificate scheme for applications within computer and communications systems. It is particularly well suited for application environments where resources such as bandwidth, computing power and storage are limited. ECQV provides a more efficient alternative to traditional certificates.

1.2 Aim

The aim of this document is to facilitate deployment of the ECQV implicit certificate scheme.

1.3 Compliance

Implementations may claim compliance with the cryptographic schemes specified in this document provided the external interface (input and output) to the schemes is equivalent to the interface specified here. Internal computations may be performed as specified here, or may be performed via an equivalent sequence of operations.

Note that this compliance definition implies that conformant implementations must perform all the cryptographic checks included in the scheme specifications in this document. This is important because the checks are essential for security.

1.4 Document Evolution

This document will be reviewed at least every five years to ensure it remains up to date with cryptographic advances.

Additional intermittent reviews may also be performed from time-to-time as deemed necessary by the Standards for Efficient Cryptography Group.

External normative standards contain provisions, which, though referenced in this document, constitute provisions of this standard. At the time of publication, the versions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent versions of the standards indicated below.

1.5 Intellectual Property

The reader's attention is called to the possibility that compliance with this document may require use of inventions covered by patent rights. By publication of this document, no position is taken with respect to the validity of claims or of any patent rights in connection therewith. The patent holder(s) may have filed with the SECG a statement of willingness to grant a license under these

rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license. Additional details may be obtained from the patent holder(s) and from the SECG web site, `www.secg.org`.

2 Cryptographic Components

This section briefly reviews the cryptographic components used in this standard (such as hash functions and random number generators). Much of the content in this section is presented in greater detail in [SEC 1, §3]. Some of the content is also available in ANSI standards, such as [ANSI X9.62].

In this document, the term *Approved* refers to a cryptographic component specified as (or within) a current SECG standard, an ANSI X9 standard, or listed in the ANSI X9 registry. The term is used to specify the requirement that primitives such as hash functions and random number generators must conform to existing standards in order to be used with ECQV.

2.1 Security Levels

Each of the various cryptographic ingredients to the ECQV scheme has a range of security levels. Security levels are measured in bits. Essentially, each extra bit of security doubles the amount of computation (required by a brute-force attack) to compromise security or, roughly equivalently, halves an adversary's probability of compromising security. At a security level of 128 bits, an adversary would require a computation of roughly 2^{127} or more bit operations to compromise security with probability greater than 50%.

This version of this Standard only recognizes Approved security levels. The currently Approved security levels are the following: 80, 112, 128, 192, and 256 bits.

2.2 Hash Functions

Hash functions are used in the ECQV scheme to compute a digest of a certificate that is being generated or verified. Hash functions are also used in verifiably random elliptic curve domain parameter generation and validation, and can also be used in random number generators.

The security level associated with a hash function depends on its application. Where collision resistance is necessary, the security level is at most half the output length (in bits) of the hash function. Where collision resistance is not necessary, the security level is at most the output length (in bits) of the hash function. Collision resistance is generally needed for computing message digests for digital signatures.

The hash functions used by the ECQV scheme shall be Approved hash functions, as Approved in the X9 Registry Item 00003, Secure Hash Standard (SHS). The security level of an Approved hash function is considered to be at most half its output length for the purposes of this Standard. The list of hash functions in [SEC 1, §3.5] are also allowed for use with this standard, excluding SHA-1. The function SHA-1 *must not* be used with this standard, with two exceptions. Use of SHA-1 is permitted to check verifiably random domain parameters from [SEC 2], when these parameters were generated using SHA-1. See Section 2 of [SEC 2] for more details. Use of SHA-1 is also permitted to implement random number generators allowed in Section 2.4, when SHA-1 is required by the standard specifying the chosen random number generator.

The hash function used for computing the certificate digest shall be a hash function whose security

level is at least the security level of the implementation of ECQV. The hash function used for verifiably random elliptic domain parameters shall be a hash function whose security level is the security level of the implementation of the ECQV scheme (with the exception of the verifiably random elliptic curve domain parameters in [SEC 2] which were generated using SHA-1).

It is expected that this standard will be revised to allow the use of SHA-3, after it is fully specified in an update to the FIPS 180 Secure Hash Standard.

2.3 Hashing to Integers Modulo n

In many steps of ECQV, an arbitrary length octet string is hashed using a cryptographic hash function H, and then converted to an integer modulo n. This section specifies the resulting function, denoted $H_n : \{0, 1, \ldots, 255\}^* \to [0, \ldots, n-1]$, which maps arbitrary length octet strings to integers modulo n.

Input

1. An arbitrary length octet string S.
2. A hash function H hashing arbitrary length octet strings to bit strings of length *hashlen*.
3. An integer n.

Actions

1. Compute $h = H(S)$, a bit string of length *hashlen* bits.
2. Derive an integer e from h as follows:
   2.1. Set E equal to the leftmost $\lfloor \log_2 n \rfloor$ bits of h.
   2.2. Convert the bit string E to an octet string E', using the Bit-String-to-Octet-String Conversion specified in [SEC 1].
   2.3. Convert the octet string E' to an integer e using the Octet-String-to-Integer Conversion specified in [SEC 1].

Output The integer e, in the interval $[0, \ldots, n-1]$.

Note that if the domain of H is limited to a maximum size, then the domain of H_n is constrained in the same way.

2.4 Random Number Generation

Within this standard, all random values must be generated using an Approved random number generator having security level at least the security level of the implementation of ECQV. Random number generators must comply with ANSI X9.82 or NIST Special Publication 800-90A [NIST 800-90A]. A suitable random number generator is also described in [SEC 1, §3.10].

2.5 Elliptic Curve Domain Parameter Generation and Validation

Elliptic curve domain parameter generation and validation shall comply with [SEC 1].

An implementation which uses both ECQV and other ECC algorithms should use a verifiably random base point G, as specified in [SEC 2].

Elliptic curve domain parameters consist of six quantities q, a, b, G, n, and h, which are:

- The field size q.
- The elliptic curve coefficients a and b.
- The base point generator G.
- The order n of the base point generator.
- The cofactor h, which is the number such that hn is the number of points on the elliptic curve.

Additionally a parameter indicating the security level is associated with the domain parameters. The security level of the domain parameters must meet or exceed the security level of the implementation of ECQV.

[SEC 2] specifies elliptic curves recommended for use with this standard. Other curves may be used, provided they can be validated using one of the methods in Section 3.1.1.2 of [SEC 1]. Domain parameters may be generated with the method given in [SEC 1], and the SEED bit string should begin with the ASCII encoding of the string "SEC4:ECQV"; the remainder of the bit string may take an arbitrary value.

3 ECQV Implicit Certificate Scheme

This section specifies the Elliptic Curve Qu-Vanstone implicit certificate scheme (ECQV).

3.1 Overview

The implicit certificate scheme is used by three entities – a Certificate Authority CA, a certificate requester U, and a certificate processor V. Requester U will obtain an implicit certificate from CA, certifying U's identity, and allowing V to obtain U's public key.[1]

The implicit certificate scheme consists of six parts. These parts will be described in detail in this section.

ECQV_Setup In this step CA establishes the elliptic curve domain parameters, a hash function, the certificate encoding format, and all parties have selected a random number generator.

[1] In the language of [RFC 2459], U is the *subject* and V is the *relying party*.

U		CA
$k_U \in_R [1, \ldots, n-1]$		
$R_U := k_U G$		
	$\xrightarrow{U,\, R_U}$	
		$k \in_R [1, \ldots, n-1]$
		$P_U := R_U + kG$
		$Cert_U := Encode(P_U, U, *)$
		$e := H_n(Cert_U)$
		$r := ek + d_{CA} \pmod n$
	$\xleftarrow{r,\, Cert_U}$	
$e := H_n(Cert_U)$		
$d_U := ek_U + r \pmod n$		
$Q_U := eP_U + Q_{CA}$		

Figure 1: Stylized description of the ECQV certificate issuance protocol.

CA generates a key pair. All parties must receive authentic copies of CA's public key and domain parameters.

Cert_Request Requester U must generate a request for a certificate, which is sent to CA. The cryptographic component of the request is a public key, generated using the same procedure as CA uses during ECQV_Setup.

Cert_Generate Upon receiving a certificate request from U, CA confirms U's identity, and creates an implicit certificate. CA sends U the response.

Cert_PK_Extraction Given an implicit certificate for user U, the domain parameters, and CA's public key, the public key extraction algorithm computes U's public key.

Cert_Reception After receiving a response to his certificate request, U ensures the validity of his implicitly certified keypair.

Certificate issuance is a two-step process, where the first step is receiving the certificate request from U, and the second step is to generate a response, containing the certificate. Figure 1 presents a stylized description of the protocol (informative), to which normative details will be added in subsequent sections.

Some differences between ECQV and traditional certificates with respect to the binding between the user's identity and public key, and the time at which this binding is confirmed by V are discussed in the commentary of Appendix B.

This standard assumes that each entity shall be bound to a unique identifier (e.g., distinguished names). This identity string is denoted `U`, `V` and `CA` for the entities U, V and CA, respectively.

3.2 Prerequisites: ECQV Setup

All parties (the certificate authority CA, the certificate owner U, and the certificate processor V) shall establish the following prerequisites in order to use ECQV.

1. CA has established a set of EC domain parameters for its use with ECQV consisting of q, a, b, G, (optionally SEED), n and h, along with an indication of the basis used if $q = 2^m$. The parameters shall have been generated with an Approved method, such as the method specified in [SEC 1, §3.1], or be Approved parameters, such as those given in [SEC 2]. CA shall have assurance of the validity of the EC domain parameters (especially if CA did not generate them himself, see [SEC 1, §3.1]). The EC domain parameters shall provide at least the desired security level, denoted s.

2. CA has selected an Approved hash function with the desired security level s for its use in the ECQV certificate generation process. (See §2.2 for more detailed requirements on the hash function.) Let H denote the hash function chosen, and let $hashlen$ denote the length of the output of the hash function. Section 2.3 describes how the output of H is converted to an integer modulo n.

3. CA and U have each chosen and initialized an Approved random number generator (as discussed in §2.4) that offers s-bit security. This random number generator shall be used for the generation of private keys created during the certificate request and certificate generation processes.

4. CA has obtained an EC key pair (d_{CA}, Q_{CA}) associated with the EC domain parameters established in Item 1 for use during certificate generation. The key pair shall have been generated using the key pair generation primitive in [SEC 1, §3.2.1], using the random number generator established in Item 3. CA shall have assurance of the validity of the key pair and assurance of the possession of the private key (see [SEC 1]).

5. The certificate owner U, and the certificate processor V have obtained, in an authentic manner, the elliptic curve domain parameters, the hash function, and Q_{CA}, CA's public key (established in Item 4). U and V shall have assurance (see [SEC 1]) of:

   5.1. the validity of the EC domain parameters,

   5.2. the validity of Q_{CA}, CA's public key, and

   5.3. possession of the private key, d_{CA} by CA.

3.2.1 Certificate Encoding Methods

A certificate encoding describes how the information to be included in the certificate should be encoded as an octet string. It also specifies what information must be included, and what information is optional, and any constraints or relationships between portions of the certificate information.

Appendix C describes three certificate encoding methods which may be used with this standard.

Fixed-length Fields is a simple, minimalist encoding which puts bandwidth efficiency before all other concerns. The certificate consists of a list of fields, each with a fixed length, and the format is shared amongst all parties. This standard requires only that the public key reconstruction data P_U be one of the fields, leaving the rest of the format open for implementors.

Minimal ASN.1 Encoding Scheme is also designed to be bandwidth-efficient. The ASN.1 specification includes basic required fields, and allows extensions. The basic fields suggested in this encoding have fixed length, and therefore may also be used with the simpler, fixed-length format.

X.509-Compliant ASN.1 Encoding is designed to include sufficient information to allow this ECQV certificate to be re-encoded as a standard X.509 certificate. Of course, since ECQV does not produce an explicit signature value, the processing logic will differ from an X.509 certificate signed with ECDSA.

3.3 Certificate Request: Cert_Request

The certificate requester U, shall use the process below to generate a certificate request.

Input

1. The elliptic curve domain parameters established by CA as determined in §3.2.
2. A string `U` representing U's identity.

Actions

1. Generate an EC key pair (k_U, R_U) associated with the established elliptic curve domain parameters using the key pair generation primitive specified in [SEC 1, §3.2.1].
2. Convert R_U to the octet string `RU` using the Elliptic-Curve-Point-to-Octet-String specified in [SEC 1, §2.3].

Output The key k_U and the certificate request $(\mathtt{U}, \mathtt{RU})$.

The value `RU` along with the purported identity `U`, make up the content of the certificate request. The value k_U is required in future steps (to compute the private key), and must be kept private. The certificate request should be sent to CA using a method preserving the data integrity of the message.

3.4 Certificate Generation Process: Cert_Generate

CA shall use the process below to generate a certificate and private key contribution data in response to a Cert_Request from U. It is assumed that CA has received `U`, and `RU` in an authenticated manner and has decided to issue a certificate.

Input

1. The elliptic curve domain parameters established by CA in §3.2.
2. The hash function H selected by CA in §3.2.
3. CA's private key d_{CA}.
4. A certificate request $(\texttt{U}, \texttt{RU})$.
5. A certificate encoding method with rules for processing as indicated in §3.2.1.
6. Additional input fields for the certificate, as described in §3.2.1.

Actions

1. Convert the octet string RU to an elliptic curve point R_U using the Octet-String-to-Elliptic-Curve-Point conversion algorithm given in [SEC 1, §2.3.4].
2. Validate R_U using the public key validation technique specified in [SEC 1, §3.2.2]. If the validation primitive outputs 'invalid', output 'invalid' and stop.
3. Generate an EC key pair (k, kG) associated with the established elliptic curve domain parameters using the key pair generation primitive specified in [SEC 1, §3.2.1].
4. Compute the elliptic curve point $P_U = R_U + kG$.
5. Convert P_U to the octet string PU using the Elliptic-Curve-Point-to-Octet-String conversion specified in [SEC 1].
6. Call the certificate encoding method with the necessary input fields and the octet string PU as indicated in §3.2.1. If the return value is 'invalid' output 'invalid' and stop, otherwise set the result as the certificate $Cert_U$.
7. Use the selected hash function to compute $e = H_n(Cert_U)$, an integer modulo n. H_n is defined in §2.3.
8. If $eP_U + Q_{CA} = \mathcal{O}$, where $\mathcal{O}$ is the identity element, return to Step 3.
9. Compute the integer $r = ek + d_{CA} \pmod{n}$.

Output $(r, Cert_U)$, where r is the private key contribution data, and $Cert_U$ is the certificate.

The response from CA may be made public. Additionally, it may be communicated over an insecure channel, as U may verify that the received information is valid, using the Cert_Reception procedure of Section 3.6. For security, the ephemeral value k must be kept private.

3.5 Certificate Public Key Extraction Process: Cert_PK_Extraction

The public key bound to the certificate is derived from the certificate using CA's public key, and is recovered using the following process. This step does not require any secret information, and may be performed by any user who knows $Cert_U$, and the public parameters output by ECQV_Setup.

Input

1. The elliptic curve domain parameters established by CA in §3.2.
2. The hash function H selected by CA in §3.2.
3. CA's public key Q_{CA} as determined in §3.2.
4. The certificate $Cert_U$.

Actions

1. Decode the certificate $Cert_U$ according to the certificate decoding methods and rules. If the return value is 'invalid' then output 'invalid' and stop, otherwise an octet string PU will be returned.
2. Convert PU to a point P_U using the Octet-String-to-Elliptic-Curve-Point conversion specified in [SEC 1, §2.3].
3. Validate P_U using the public key validation technique specified in [SEC 1, §3.2.2]. If the validation primitive outputs 'invalid', output 'invalid' and stop.
4. Use the selected hash function to compute $e = H_n(Cert_U)$, an integer modulo n. H_n is defined in §2.3.
5. Compute the point $Q_U = eP_U + Q_{CA}$.

Output Either 'invalid' or a public key Q_U.

Note that if Cert_PK_Extraction outputs Q_U, then Q_U is valid in the sense of [SEC 1]; it is a point of order n on the input EC domain parameters.

3.6 Processing the Response to a Cert_Request: Cert_Reception

This routine validates the contents of an implicit certificate and the private key contribution data issued by CA. The output of Cert_PK_Extraction, and the private key computed here, form a key pair. Recall that Cert_PK_Extraction performs the public key validity check from [SEC 1], so it is not repeated here. The certificate requester U shall use the process below, upon receipt of their certificate request response, to compute the private key for the public key output by Cert_PK_Extraction, and validate the key pair.

Input

1. The elliptic curve domain parameters established by CA in §3.2.
2. The hash function H selected by CA in §3.2.
3. The private value k_U generated by U in §3.3.
4. The output of Cert_Generate: the certificate $Cert_U$ and the private key contribution data, an integer r.

Actions

1. Compute the public key Q_U using Cert_PK_Extraction (or equivalent computations).
2. Use the selected hash function to compute $e = H_n(Cert_U)$, an integer modulo n.
3. Compute the private key $d_U = r + ek_U \pmod{n}$.
4. Compute $Q'_U = d_U G$.

Output 'valid' and d_U if Q_U is equal to Q'_U, and 'invalid' otherwise.

3.7 ECQV Self-Signed Certificate Generation Scheme

This section specifies the scheme for generating self-signed implicit certificates. In the self-signed certificate generation scheme, the user U generates the certificate request and performs the actions of CA as well, but sets CA's key pair to $(0, \mathcal{O})$ (the private key is zero and the public key is the identity element of the elliptic curve group). The certificate must indicate that it is a self-signed certificate to allow the public key to be extracted correctly.

U shall execute the following steps to create a self-signed implicit certificate.

Input

1. The elliptic curve domain parameters, as determined in §3.2.
2. The hash function H, as determined in §3.2.
3. A certificate encoding method with rules for processing as indicated in §3.2.1.
4. Additional input fields for the certificate, as described in §3.2.1.

Actions

1. Use the key pair generation primitive specified in [SEC 1, §3.2.1] to generate a key pair (k_U, P_U) associated with the established domain parameters.
2. Convert the elliptic curve point P_U to the octet string `PU` using the Elliptic-Curve-Point-to-Octet-String conversion algorithm given in [SEC 1, §2.3].
3. Call the certificate encoding method with the necessary input fields and the octet string `PU` as indicated in §3.2.1. If the return value is 'invalid' output 'invalid' and stop, otherwise set the result as the certificate $Cert_U$. $Cert_U$ must indicate that it is self-signed.
4. Use the selected hash function to compute $e = H_n(Cert_U)$, an integer modulo n.
5. Compute the private key $d_U = ek_U \pmod{n}$.

Output If any of the above verifications has failed, then output 'invalid' and stop; otherwise, output 'valid', $Cert_U$ as U's self-signed implicit certificate, and d_U as the corresponding private key.

3.8 ECQV Self-Signed Implicit Certificate Public Key Extraction

This section specifies the scheme for extracting the public key from a self-signed implicit certificate. As with the process for generating a self-signed certificate, extracting the public key follows the extraction process but with the CA key pair set to $(0, \mathcal{O})$.

Input

1. The elliptic curve domain parameters, as determined in §3.2.
2. The hash function H, as determined in §3.2.
3. A certificate encoding method with rules for processing as indicated in §3.2.1.
4. The self-signed certificate $Cert_U$ output by the process in §3.7.

Actions

1. Decode the certificate $Cert_U$ according to the certificate decoding methods and rules. If the return value is 'invalid' then output 'invalid' and stop, otherwise an octet string `PU` will be returned. Ensure that $Cert_U$ is self-signed, by checking the appropriate field. If $Cert_U$ is not self-signed, output 'invalid' and stop.
2. Convert `PU` to a point P_U using the Octet-String-to-Elliptic-Curve-Point conversion specified in [SEC 1, §2.3].

3. Validate P_U using the public key validation technique specified in [SEC 1, §3.2.2]. If the validation primitive outputs 'invalid', output 'invalid' and stop.

4. Use the selected hash function to compute $e = H_n(Cert_U)$, an integer modulo n.

5. Compute the point $Q_U = eP_U$.

Output: If none of the above steps has output 'invalid' output the point Q_U as the public key corresponding to $Cert_U$. Otherwise output 'invalid'.

A Glossary

This section provides a glossary of the terms, acronyms, and notation used in this document.

Please refer to the glossary of SEC1 [SEC 1] for any term, acronym, or notation not specified in this section.

A.1 Terms

Terms used in this document include:

(traditional) certificate	Information including the public key and identity of an entity, cryptographically signed by a Certificate Authority. See "certificate" in Section A.1 of SEC1 [SEC 1].
implicit certificate	Information including public-key reconstruction data and the identity of an entity that constitute the certificate of that entity.
public-key reconstruction data	An elliptic curve point contained in the implicit certificate, from which any party with access to *CA*'s public key can reconstruct the public key associated with the certificate.
private-key reconstruction data	Value computed by a *CA* during the creation of an implicit certificate. This may be a public value that allows the certificate requester to compute his private key. Also called the *CA contribution* to the private key.
to-be-signed-certificate data	Data to be included in a certificate or implicit certificate. This data includes the identity of the certified entity, but may also include other data, such as the intended use of the public key, the serial number of the certificate, and the validity period of the certificate. The exact form of this data depends on the certificate encoding being used, and is selected by the CA.

A.2 Acronyms

The acronyms used in this document denote:

ECQV	Short for the Elliptic Curve Qu-Vanstone implicit certificate scheme described in Section 3.
CA	The certificate authority.

A.3 Notation

The notation adopted in this document is:

$Cert_U$	Implicit certificate for user U.
P_U	Public reconstruction data for user U, an elliptic curve point.
`PU`	Public reconstruction data for user U, encoded as an octet string.
I_U	To-be-signed-certificate data, non-cryptographic information about user U, such as their identity and the validity of the certificate.
R_U	Certificate request value created by U, an elliptic curve point.
`RU`	Certificate request value created by U, encoded as an octet string.
Q_U	User U's public key.
Q_{CA}	CA's public key.

B Commentary

The aim of this section is to supply implementers with relevant guidance. However, this section does not attempt to provide exhaustive information but rather focuses on giving basic information and including pointers to references which contain additional material. Furthermore, this section concentrates on supplying information specific to implicit certificates. Extended commentary on ECC in general – addressing issues like parameter selection and implementation of elliptic curve arithmetic – can be found in [SEC 1, Appendix B]. This section provides a commentary on the ECQV implicit certificate scheme. It discusses properties specific to this scheme, and some security considerations specific to ECQV certificates.

Binding of Identity and Key Pair With all digital certificates that bind an identity to a public key, there are two aspects of certification to consider. The first is the binding between the user identity and his public key. The second is assurance that the user has knowledge of his private key.

The implicit certificate generation algorithm yields a static public key purportedly bound to U (i.e., purportedly issued by CA). Confirmation that this public key is genuinely bound to U is only obtained after use of the corresponding key pair (e.g., via execution of an authenticated key agreement scheme involving this key pair). Thus, with implicit certificates, the binding of an entity and its public key and knowledge of the private key are verified in unison, during key usage.

This situation differs from ordinary certificates (e.g., X.509 certificates), where the binding between U and his public key is confirmed by verifying CA's signature in the certificate. Proof that U knows the corresponding private key is only obtained during cryptographic usage of the key pair. Some certificate issuance protocols require U to prove knowledge of his private key to CA. In this case, another user V has indirect assurance that U knows his private key if the certificate is valid (since V trusts CA to perform this check when the certificate was issued).

Key Pair Generation With traditional certificates, key pair generation and certificate issuance are two independent processes. A user can present an arbitrary public key to a CA for certification. In ECQV the situation is somewhat different. When a user requests an implicit certificate for a public key from a CA, this public key (and the private key) is a randomized result of the joint computation by the user and the CA.

This has the direct consequence that once an ECQV implicit certificate is issued, one cannot get another ECQV implicit certificate for the same public key from a different CA. It is possible however, to obtain a traditional certificate on an ECQV public key.

Domain Parameters Associated with a Key Pair Recall that U's public key associated with an ECQV implicit certificate $Cert_U$ is computed as $Q_U = H_n(Cert_U) \cdot P_U + Q_{CA}$. Since Q_{CA} is defined over a certain elliptic curve, this computation must be performed over the same curve. This means that the constructed user's public key is defined over the same elliptic curve as CA's public key. Hence, the key pair certified by an ECQV implicit certificate has to be defined over the same elliptic curve parameters as used by CA. In particular the security level of the certified

key pair is the same as CA's key pair.

Key-Certificate Confirmation In some applications the subject may need to periodically confirm that the private key and the public key reconstructed from a certificate correspond. This check is done during certificate reception, but it may also be required later. For example, if certificates are requested in batches by a device manufacturer, and subsequently injected into devices (along with the private key), the device should check that the injected key and certificate are correct.

If such a confirmation functionality is desired, it is recommended that implementations provide a routine similar to the example below. Additional checks on the certificate data may also be performed at this time.

```
boolean KeyCertConfirmation(secret_key sk, implicit_cert CertU){
    public_key QU = cert_PK_Extraction(CertU);
    if( sk*G == QU)
        return true;
    else
        return false;
 }
```

Efficiency An advantage of implicit certificates is that, since they contain only the public reconstruction data instead of the subject's public key and the CA's signature, they may be smaller than traditional certificates. We specify three certificate formats in Appendix C.

With respect to computational efficiency, we note that the public key extraction step may be combined with other operations. For instance, if a protocol requires computation of zQ_U where Q_U is a public key implicitly certified by $Cert_U$ then instead of computing $Q_U = H(Cert_U)P_U + Q_{CA}$ followed by zQ_U (two scalar multiplications) it may be faster to compute $zH(Cert_U)P_U + zQ_{CA}$ (using fast scalar multiplication/exponentiation techniques).

Demonstrating Knowledge of the Secret Key During Certificate Issuance When an entity U requests a traditional certificate for a public key, U should prove to the CA it knows the corresponding private key. This is to prevent U from choosing an arbitrary public key, that may already belong to another user, and have it certified. This situation is clearly undesirable (and may even lead to security problems).

With implicit certificates this proof is unnecessary, as there is no public key before the certificate is issued. Further, U has no control over the final value of his public key, due to the CA's contribution, making it impossible for U to cause the confusion described above.

Unforgeability Unlike traditional certificates, an implicit certificate does not contain a digital signature. In fact, one could simply choose an arbitrary identity I and a random value to form a certificate. Together with the public key of a CA, this generates a public key for the entity identified by I. However, if one constructs an implicit certificate in such a way, i.e., without interacting with

the CA, it is infeasible to compute the private key that corresponds to the public key generated by the certificate. See [BGV01] for a security analysis of ECQV.

Another difference between traditional certificates and implicit certificates is that when presented with a valid traditional certificate, one knows that the certificate belongs to *someone*. A valid certificate containing the certificate data string I_U is a proof that the CA signed this certificate for U, and also that U knows the private key corresponding to the public key included in the certificate. One does not have this guarantee with implicit certificates. It is trivially possible to construct an implicit certificate $Cert_U$ such that the private key corresponding to the public key computed as $Q_U = H_n(Cert_U) \cdot P_U + Q_{CA}$ is unknown.

This fact suggests a denial-of-service type attack, where a party V is flooded with protocol requests using "fake" implicit certificates. The fact that the private key of the fake certificate is unknown is only revealed after V has performed most of the protocol. Of course, a similar attack can be launched in a system using traditional certificates. In this case, the attacker would flood a party with various certificates belonging to other entities. The certificates are valid, but the attacker does not know the private key of the corresponding public key.

Composability Composition of ECQV with other primitives requires care. By *composition*, we mean using the implicit public key computed with Cert_PK_Extraction with another cryptographic primitive. For instance, using a validated ECQV public key as an ECDSA verification key is known to be secure, provided the signed message is not the same as the certificate, i.e., verifiers should not accept a signature from U on the message $Cert_U$. See the analysis of ECQV-certified ECDSA by Brown et al. [BCV09]. Composition of ECQV with other primitives from SEC1 such as key-agreement (ECMQV) and encryption (ECIES) have not been formally analyzed in the literature, however no attacks against these compositions are known either.

Certificate Chains Another type of composition of ECQV is with itself, in a certificate chain. Certificate chains are often used in public-key infrastructure to form hierarchical certification authorities. In a chain, an ECQV key pair (belonging to a CA or sub-CA) is used to issue ECQV certificates (to a sub-CA or end entity).

The following outlines an attack against ECQV chains of length four or more, or chains of length three composed with a digital signature, is possible. The formulae for reconstructing the public keys from each certificate can combined into a single formula for reconstructing the final public key in the chain. An equation can then be formed by equating this combined formula to the formula for the public in terms of the private key. Or, when one wishes to verify a signature, the signature verification equation can be included in the combined equation instead. The problem of finding a solution to the combined equation can be interpreted as instance of Wagner's generalized birthday problem.

Recall that the generalized birthday problem is to find a sequence of independent pseudorandom group elements summing to zero. The case of a sum of two values can interpreted as finding a collision in a pseudorandom function. One might expect the fastest algorithm to be exhaustive search. The collision problem can be solved faster than one might expect because of the birthday surprise paradox.

Wagner's tree algorithm solves the generalized birthday problem when the group has certain additional structure, such as ability to compare elements that are preserved under the group operation. As with collision search algorithm, Wagner's tree algorithm is faster than exhaustive search. In fact, as the number of terms in the sum increase, Wagner's algorithm becomes even faster.

Wagner's tree algorithm can be applied to forge certificate chains where the private key of the last certificate is known by the attacker, or to forge a signature signed by the end entity of a certificate chain. In this latter case the private key need not be known. In both these cases the verifying equation can be viewed as equations in the private key space, which is an ordinary modular group with ordering preserved under the group operation.

When either the length of the certificate chain is four, or when the certificate chain is three and a signature by the last entity named in the chain is to be forged, Wagner's algorithm is faster than solving the discrete logarithm problem directly. Specifically, in this case Wagner's algorithm takes about $\sqrt[3]{n}$ group operations where n is the order of the group, whereas the best known algorithms for solving the discrete logarithm take about $\sqrt{n}$ group operations.

A more detailed description of this attack will be published and made available on the SECG website.

Wagner's tree algorithm is rather sensitive to the conditions of the underlying problem, such as the independence of the terms in the sum. This leads to several choices of additional actions likely to prevent this certificate chain forgery attack. Further research is needed to assess the security of a preferred mitigation suitable for standardization. Ideally, this research will lead to security proofs of chaining modes for ECQV.

If implementers utilize certificate chains of length three or more it is recommended that they provide counter measures that thwart these generalized birthday attacks.

C Representation of ECQV Certificate Structures

Here we specify the encoding and decoding operation primitives that may be used to generate ECQV certificates and process ECQV certificates. We provide three options, a simple fixed-length encoding, a minimal ASN.1 encoding, and an X.509-like ASN.1 encoding. Implicit certificates are specifically defined to reduce bandwidth requirements, therefore creating highly extensible and verbose encoding methods are counterproductive to the goals of the scheme. The schemes defined indicate rules associated with elements of the certificate. These rules typically enforce public key infrastructure policies, such as key-usage, validity periods, issuer identifier, subject identifier, etc.

C.1 Fixed-Length Fields

The fixed-length fixed-fields encoding and decoding methods uses concatenation of a fixed and agreed number of octet string fields of fixed length. We do not specify fields to be included. This is left to individual implementations. The fixed-length fixed-fields encoding may easily be customized to a particular application to include only those fields which are necessary for the application, leading to a compact certificate. The fields given in the minimal encoding scheme of §C.2 may also be used with this encoding, as they have fixed length.

Prerequisites

The entities using this method shall agree on the following choices for the method:

- A number of fields f.
- The octet length of each field, denoted len_i for $i = 1$ to f.
- The index i, of the field that will contain the encoded public key reconstruction data (an octet encoded elliptic curve point).
- Validity rules for each of the field elements.

Fixed-Length Fixed-Fields Encoding Operation

Input The input to the encoding operation is:

- Octet strings $F_1, F_2, \ldots, F_f$,
- $F_i = \texttt{PU}$ an octet encoded elliptic curve point for some i in $[1, \ldots, f]$.

Actions Compute the ECQV certificate as follows:

- Verify each field F_i is of length len_i and if not return 'invalid' and stop.

- Validate that each field satisfies the rules specified in the prerequisites, and if any fail return 'invalid' and stop.
- Create the certificate $Cert = F_1||F_2||\ldots||F_f$.

Output The octet string $Cert$ or 'invalid.'

Fixed-Length Fixed-Fields Decoding Operation

Input The input to the decoding operation is:

- The certificate $Cert$.

Actions Decode the certificate as follows:

- Verify that the length of $Cert$ is equal to $len_1 + len_2 + \ldots + len_f$, if not return 'invalid' and stop.
- Parse the certificate into pre-defined length segments $Cert = F_1||F_2||\ldots||F_f$.
- Validate that each field satisfies the rules specified in the prerequisites, and if any fail return 'invalid' and stop.

Output The field values $F_1, \ldots F_f$, where one of F_i = `PU`, an octet-encoded public key reconstruction value.

C.2 ASN.1 Encodings: Minimal and X.509-Compliant

We provide two ASN.1 encodings. The first is called the minimal encoding scheme (MES). The MES is designed to be as compact as possible, and is a list of basic certificate fields which are common to many applications. The fields in the MES are fixed length, which means the fields in this format may be encoded using the simple encoding of §C.1, however, for greater portability it may also be encoded with the ASN.1 syntax provided in this section.

The second format we describe is an X.509 compliant format. The X.509 certificate format is given in [RFC 2459]. This format is larger, but allows ECQV certificates to be parsed as X.509 certificates. We also give a mapping between the two formats which allows a MES encoded ECQV certificate to be re-encoded as an X.509 encoded ECQV certificate. Re-encoding in the other direction is possible, however, since the X.509 format allows more information to be stored in the certificate, some information may be lost when re-encoding. Put another way, the mapping of ECQV certificates from MES to X.509 encodings is one to many, because of the additional fields in the X.509 encoding. The ASN.1 module specifying the encodings of ECQV certificates is given below.

In both ASN.1 encoded formats, to indicate that a certificate is self-signed, the issuerID (or issuer) field should be set to zero.

```
-- ---------------
-- ECQV certificate format: minimal encoding scheme (MES)
-- ASN.1 is used to describe the format, but these fields
-- could also be used with the fixed-length field encoding.
-- The times are represented using UNIX time, i.e., # of seconds
-- since the unix epoch: http://en.wikipedia.org/wiki/Unix_time
-- The validFrom field uses 40-bit values to avoid problems in
-- 2038 (when 32-bit values won't be enough).
-- ---------------

ANSI-X9-YY{iso(1) member-body(2) us(840) 10045 module(0) 2}

DEFINITIONS EXPLICIT TAGS ::=
BEGIN

ansi-X9-YY OBJECT-IDENTIFIER ::= {iso(1) member-body(2) us(840) 10045}

ECQVCertificate  ::=  SEQUENCE  {
    type MESType DEFAULT t1,-- one byte, see below
    serialNumber    OCTET STRING (SIZE (8)),
    curve Curve,-- named curve, see below
    hash Hash,
    issuerID  OCTET STRING (SIZE (8)),
    validFrom   OCTET STRING (SIZE (5)),-- 40-bit Unix time
    validDuration   OCTET STRING (SIZE (4)),-- 32-bit # of seconds
    subjectID  OCTET STRING (SIZE (8)),
    usage KeyUsage,  -- one byte, described below
    pubKey OCTET STRING,
    pathLenConstraint INTEGER (0..255) OPTIONAL,
    ...,
    -- Extensions:
    algorithm[1]    AlgorithmIdentifier OPTIONAL,
    email[2] IA5String (SIZE (0..128)) OPTIONAL
}

-- Notes:
-- * The 32-bit # of seconds allows certs to be valid for up to 136 years. If
--   validDuration = 2^32 -1 (the maximal unsigned 32-bit integer) then the
--   certificate is valid forever, i.e., it has no expiry
-- * The subjectID could be a MAC address (they are 48-bit values, and the
--   OUI is the least significant three octets)
-- * The issuerID is an 8 byte identifier for a CA, assumed to be associated
--   with the CA's public key by subjects and relying parties out of band.
-- * If extensions are used, type MUST be t2 and the extensions algorithm and
--   email MUST be present.
-- * The size of pubKey varies depending on the curve chosen.  Note this is only
--   the public key reconstruction data.
-- * The algorithm extension is used to specify what the user's public key will
```

```
--   be used for (e.g., ecdsa-with-sha256)
-- * The email extension can be null terminated instead of fixed length, and be
--   a maximum of 128 bytes.
-- * The total size of a type 1 certificate is: 37 bytes + size of the public key
--   reconstruction value

-- The type of the certificate indicates whether the certificate contains
-- extensions.  There are two possibilities:
--  Type 2: |required fields|extensions|
--  Type 1: |required fields|
MESType  ::=  INTEGER  {
    t1(0), -- type 1: no extension(s),
    t2(1), -- type 2: with extension(s)
}

-- Curves for use with ECQV. Includes all curves listed in SEC2. See Section 2.5.
Curve  ::=  INTEGER  {  secp192k1(0), secp192r1(1), secp224k1(2),
secp224r1(3), secp256k1(4), secp256r1(5), secp384r1(6), secp512r1(7),
sect163k1(8), sect163r1(9), sect233k1(10), sect233r1(11), sect239k1(12),
sect283k1(13), sect283r1(14), sect409k1(15), sect409r1(16), sect571k1(17),
sect571r1(18)  }

Hash  ::=  INTEGER  {  id-sha224(0), id-sha256(1), id-sha384(2),
id-sha512(3) }

-- The KeyUsage bit string is a bit field, where bits are asserted
-- in indicate valid uses of the key, as in RFC 5280.
-- See RFC 5280, Section 4.2.1.3 for more details.
KeyUsage ::= BIT STRING {
digitalSignature        (0),
        nonRepudiation          (1),
        keyEncipherment         (2),
        dataEncipherment        (3),
        keyAgreement            (4),
        keyCertSign             (5),
        cRLSign                 (6),
-- the last bit in the byte is always zero (7)
}

-- ---------------
-- X.509 ECQV certificate structures
-- ---------------
-- This certificate format is compliant with x.509, and corresponds to a MES
-- encoded ECQV certificate The descripition of X.509 is here:
--  http://www.ietf.org/rfc/rfc5280.txt
-- ---------------

-- OIDs required: ecqv-with-sha224, ecqv-with-sha256, ecqv-with-sha384,
```

```
-- ecqv-with-sha512, ecqv-with-aes-mmo

-- signatureAlgorithm is the CA's signing algorithm (always ECQV-something).
ECQV-X509-Certificate  ::=  SEQUENCE  {
    tbsCertificate     TBSCertificate,
    signatureAlgorithm AlgorithmIdentifier,
}

-- This is the certificate data to be signed.  Notes:
-- * versions have the same interpretation as in rfc5280
-- * signature denotes the CA's signing algorithm. MUST be the same as
--   signatureAlgorithm above.  The valid algorithm identifiers are the
--   OIDs listed above.
-- * subjectPublicKeyInfo contains the public key reconstruction data
-- * extensions can be any valid X.509 extension. If present, version
--   MUST be v3
TBSCertificate  ::=  SEQUENCE  {
    version         Version DEFAULT v1,
    serialNumber    OCTET STRING,
    signature       AlgorithmIdentifier,

    issuer          Name,-- corresponds to issuerID in minimal encoding
    validity        Validity,-- as in rfc5280 (X.509)
    subject         Name,-- corresponds to subjectID in minimal encoding
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    extensions[3]   Extensions OPTIONAL
}

Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }


CertificateSerialNumber  ::=  INTEGER

Validity ::= SEQUENCE {
     notBefore      Time,
     notAfter       Time
}


-- This is the way time is done in PKIX (from rfc 5280). Some details of the
-- time conversion are given in the MES->X.509 mapping below.
Time ::= CHOICE {
     utcTime        UTCTime,
     generalTime    GeneralizedTime
}

UniqueIdentifier  ::=  BIT STRING
```

```
-- The AlgorithmIdentifier specifies the algorithm the user will use their
-- keypair for.  Note the parameters MUST be empty, since the user's key
-- inherits the paramters from the CA (No equivalent in minimal encoding.)
-- Note that subjectPublicKey is the public key reconstruction data P_U,
-- NOT the public key.
SubjectPublicKeyInfo  ::=  SEQUENCE  {
     algorithm            AlgorithmIdentifier,
     subjectPublicKey     ECPoint
}

Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::=  SEQUENCE  {
     extnID      OBJECT IDENTIFIER,
     critical    BOOLEAN DEFAULT FALSE,
     extnValue   OCTET STRING
}

-- This is similar to the x.509 ASN.1 encoding of the algorithm field,
-- but drops the parameters field, since the subjects' public key
-- can only be used with the same parameters as the CA.
AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm          OBJECT IDENTIFIER,
}

-- The following elliptic curve structures are included for completeness.

-- Named Elliptic Curves in ANSI X9.62.
ansi-X9-62 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) 10045 }
ellipticCurve OBJECT IDENTIFIER ::= { ansi-X9-62 curves(3) }

-- Elliptic Curve parameters may be specified explicitly, specified implicitly
-- through a "named curve", or inherited from the CA.  Note that explicitly
-- specified parameters are not supported by the MES therefore, X.509 ECQV
-- certificates with explicitly specified parameters cannot be encoded using the
-- MES format.

EcpkParameters ::= CHOICE {
    ecParameters  ECParameters,
    namedCurve    OBJECT IDENTIFIER,
    implicitlyCA  NULL
}

-- Elliptic curve parameters
ECParameters  ::= SEQUENCE {
    version   ECPVer,
    fieldID   FieldID,
    curve     Curve,
```

```
    base      ECPoint,    -- Base point G
    order     INTEGER,                -- Order n of the base point
      cofactor  INTEGER  OPTIONAL      -- The integer h = #E(Fq)/n
   }

FieldID ::= SEQUENCE {
    fieldType   OBJECT IDENTIFIER,
    parameters  ANY DEFINED BY fieldType
}

ECPVer ::= INTEGER {ecpVer1(1)}
ECPoint ::= OCTET STRING


------------------
-- X.501 Name type
------------------
Name ::= CHOICE {
    rdnSeq RDNSequence
}

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::=
    SET OF AttributeTypeAndValue

AttributeTypeAndValue ::= SEQUENCE {
    type     OBJECT IDENTIFIER,
    value    [0] EXPLICIT ANY DEFINED BY type
}

DirectoryString ::= CHOICE {
    teletexString TeletexString (SIZE (1..MAX)),
    printableString     PrintableString (SIZE (1..MAX)),
    universalString     UniversalString (SIZE (1..MAX)),
    utf8String          UTF8String (SIZE (1..MAX)),
    bmpString           BMPString (SIZE (1..MAX))
}

-- --------------------------------
-- MES and X.509 field equivalence
-- Here we describe how to encode a MES encoded certificate as an X.509
-- certificate (and vice-versa, the mapping given here is 1:1. Of course, the
-- X.509 may contain additional fields.) Field by field correspondence is given.
-- Note there is no loss of time precision since X.509 times MUST include
-- seconds and MUST NOT include fractional seconds.
-- --------------------------------
```

```
-- MES field        X.509 field
-- ---------        -----------

-- type         tbsCertificate.version = v3
-- Note: version must always be v3, since the KeyUsage field is a basic field
-- in MES, but extension in X.509

-- serialNumber   tbsCertificate.serialNumber

-- curve          tbsCertificate.signature.ecParams.curve

-- hash           given by the name of the AlgorithmIdentifier
--                tbsCertificate.signature

-- issuerID       tbsCertificate.issuer (note it uses the Name type)

-- validFrom      tbsCertificate.Validity
-- validDuration
-- Note: Validity.notBefore = validFrom,
-- Validity.notAfter = notBefore + validDuration.
-- Also note: MES encodes time as unix time, and X.509 uses the Time type.
-- If validDuration = 2^32 -1 (the maximal unsigned 32-bit integer) then the
-- certificate is valid forever, i.e., it has no expiry.

-- subjectID      tbsCertificate.subject (note it uses the Name type)

-- usage          tbsCertificate.usage (Note: this is an X.509 extension)

-- pubKey         tbsCertificate.subjectPublicKeyInfo.subjectPublicKey

-- ***            signatureAlgorithm = ecqv-with-hash
-- Note: both signatureAlgorithm and tbsCertificate.signature should be the same.

-- algorithm      subjectPublicKeyInfo.algorithmIdentifier

-- email          X.509 email extension (version must be v3)


END
```

D References

[ANSI X9.62] *ANS X9.62:2005: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 2005.

[NIST 800-90A] E. Barker and J. Kelsey. *SP 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. NIST, Jan. 2012. http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf.

[RFC 2459] R. Housley, W. Ford, W. Polk and D. Solo. *RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile*. IETF, 1999. www.ietf.org/rfc/rfc2459.txt.

[SEC 1] SECG. *SEC 1: Elliptic Curve Cryptography*, May 2009. Version 2.0. www.secg.org.

[SEC 2] ———. *SEC 2: Recommended Elliptic Curve Domain Parameters*, Jan. 2010. Version 2.0. www.secg.org.

[BCV09] D. Brown, M. Campagna and S. Vanstone. *Security of ECQV-certified ECDSA against passive adversaries*. Cryptology ePrint Archive Report 2009/620, IACR, 2009.

[BGV01] D. Brown, R. Gallant and S. Vanstone. *Provably secure implicit certificate schemes*. In P. F. Syverson (ed.), *Financial Cryptography — FC 2001*, LNCS 2339, pp. 156–165. Springer, Feb. 2001.