

## Public Key Validation: A Piece of the PKI Puzzle

Submission to IEEE P1363

March 10, 2000

Don B. Johnson

[djohnson@certicom.com](mailto:djohnson@certicom.com)

### Abstract

Public Key Validation (PKV) consists of arithmetic tests that help ensure that the components of a candidate public key conform to the key generation requirements of a standard. If an invalid public key is used, any intended security may be void. Each user in the Public Key Infrastructure (PKI) is responsible to decide whether to seek assurance regarding the validity of a candidate public key or to accept the risks of using an possibly invalid public key. A CA may provide this assurance to its clients as part of its public key certification process.

### The Players

Alice and Bob are typical users of the Public Key Infrastructure (PKI). Eve is an adversary trying to gain an unfair advantage, commit fraud, etc. Dim is a typical user of the PKI also, but is perhaps too trusting. The Certificate Authority (CA) is a trusted third party that issues a certificate that binds the identity of individuals with their public key; in doing this, the CA has issued a Certificate Practice Statement (CPS) that states which checks that the CA considers appropriate (based on the intended use of the public key) to perform and any liability that the CA may assume for its mistakes.

### Signature Scenario

Eve obtains a public key certificate from a CA, signs a message and sends the certificate and signed message to Bob. Bob verifies that Eve's certificate is not revoked and verifies the digital signature. The reader should recognize this as the normal PKI signature scenario. One question to ask is, "Does Bob know the message is from Eve?" The answer one wishes is: "Yes, Bob knows the message is from Eve."

However, if the public key in Eve's certificate does not conform to the specifications of the appropriate cryptographic standard (for whatever reason), it may simply not represent an instantiation of a hard problem in mathematics. For example, when using RSA, the public modulus is supposed to be the product of two large primes and the (hopefully) hard problem facing an adversary trying to break an RSA key is trying to factor the modulus. Factoring the modulus then allows determination of the corresponding RSA private key. But (as a "toy" example) what if the RSA modulus was a prime instead? In this case, a prime is trivial to factor and therefore it is trivial for anyone to determine the corresponding "private" key. Yet this "RSA" public key with a prime modulus would appear to work in any implemented RSA public key calculation (that is, the arithmetic

calculation using the invalid public key would produce a result and would not abort).

There are many more subtle examples than the “toy” example given above. For example, what if a bit-flip in the modulus caused a candidate 1024-bit RSA key to be the product of primes all less than 100 bits, such a key could be factored with a feasible amount of work. It is not the purpose of this paper to examine each and every possible flaw or try to assess how likely each possible flaw might be. Rather, for the purposes of this paper it is enough for the reader to realize that a candidate public key may or may not be valid, that is, it may or may not conform to the key generation requirements as specified in some standard. Note that use of methods that are “provably secure” under some model (such as the Random Oracle model) may not be secure if an invalid key is used.

What is the Problem?

One way to look at this concern is that each component of a public key is supposed to have a certain arithmetic structure. Each component should be valid both (A) considered by itself, (e.g., in the NIST Digital Signature Algorithm (DSA)  $q$  is a 160-bit prime) and (B) in relation to other components (e.g., in the DSA,  $q$  (besides being prime) is also supposed to be a factor of  $(p-1)$ ). However, a candidate public key just appears as a “bag of bits”, for example, in an ASN.1 structure in an X.509 certificate. There are many more bitstrings that do not represent a valid public key than there are bitstrings that do. How does one know that any particular “bag of bits” actually conforms to the requirements for a public key? The simply and straightforward answer is that unless someone looks at the public key and does some checking, no one knows for sure.

How Might an Invalid Public Key Exist?

Given that it is conceivable for a public key to be invalid, how might such a beast come to exist. One reason could be that an undetected error occurred during the asymmetric key pair generation process. This should be expected to be rare, but the probabilities depend on the exact nature of the error and the exact details of an implementation (including any potential coding errors that might appear only in certain rare circumstances). It appears that cosmic rays, voltage fluctuations, temperature extremes, physical contortions and the like can sometimes cause bit flips in a physical implementation. A common engineering practice is to “check calculations for mistakes.” A cyclic redundancy check (CRC) is often used to detect transmission errors. The plain scientific fact is that bit errors sometimes occur. The question is: what, if anything, does one do about this possibility?

One way to view PKV is that PKV does arithmetic validation tests on a candidate public key to check for possible key generation calculation errors. As the PKV tests are public, they can be audited or an auditor can do the PKV tests himself. Use of PKV in this manner can be thought of as similar to using a CRC to detect transmission errors. That is, PKV can be used following the “good engineering practice” philosophy.

However, there is another possible reason why an invalid public key might exist. As people like Eve (that is, an adversary trying to exploit the system) exist, there is also the possibility of a deliberate introduction of an invalid public key, either by Eve herself when talking to Alice or by Eve trying to insert an invalid public key into a communication between Alice and Bob. In doing this, Eve is trying to obtain some illicit advantage, such as being able to read mail between Alice and Bob, defrauding Alice, etc. Use of PKV in this instance can be thought of as similar to using other cryptographic techniques to achieve an end result. In this case, one uses PKV to detect a deliberate error in a candidate public key that, if undetected, may lead to undesirable results, such as loss of intended security.

In summary, an invalid public key might exist due to a deliberate act or due to an honest mistake. Let us return to the signature scenario and examine two related versions of what might happen. Bob ensured that Alice's certificate was not revoked and that the signature on Alice's message verified. Alice knows she did not sign the message, so she knows something went wrong. Alice assumes that an adversary somehow guessed her private key. So Alice revokes her public key, generates a new key pair, and gets her new public key certified.

However, what about the non-repudiation feature of the signature? Bob did what he was supposed to do (at least in the current paradigm). But Alice wants to repudiate the signature as she knows she did not sign it (e.g., assume the message was for a significant amount of money).

Alice looks at things closer and discovers that somehow there was an error during key pair generation. This starts to look a little messy now, as Alice is considering holding the manufacturer liable for damages. In any case, Alice goes before a judge and shows the judge that the supposed public key did not actually represent a mathematical problem that was difficult to break. At this point it seems clear that there are many ways this version of the scenario could end. Perhaps Alice is required to accept the signature, perhaps Alice is allowed to repudiate, perhaps the vendor is held responsible, perhaps the CA covers the loss as part of insurance, etc.

None of these possible outcomes seem very desirable. What Alice, Bob, the vendor, the CA, and the judge really wish is that the error was detected before the invalid public key ever was used. That way, there is no signature repudiation mess to untangle, Alice simply generates a new key pair.

Now instead of Alice doing this, assume Eve did it. For example, suppose Eve somehow discovers that if she exposes her cryptosystem to an intense UV light (for example) during key generation, there is a 1% chance of a bit flip. This system weakness was not discovered during system test, as normal amounts of UV radiation did not cause any errors. Eve generates 100 different key pairs, sees which one has the error and gets that one certified. Now she just pretends

to be innocent Alice in the about scenario. So the question is: “How does anyone (let alone a judge) tell Alice apart from Eve?” That is, how does one discriminate between a situation where an honest mistake occurred and one where a deliberate act resulted in an error? This seems to be very difficult to accomplish if it is able to be done at all. In a PKI, we really do not want to lump honest Alice with dishonest Eve and say we cannot tell them apart.

Enter PKV. If Bob does PKV before accepting the signature, then Bob rejects the candidate public key and therefore the signature. Honest Alice is told something is wrong with her key, so she revokes it and generates a new one without errors. Scheming Eve is also told about the error and Bob is not duped by Eve into accepting a message that Eve will later try to repudiate. The principle being used in this situation is simple: Whenever one finds oneself in a situation with potential ambiguity, try to remove the ambiguity.

Rather than having Bob do PKV himself, it is natural for Bob to ask his CA to do it for him, as Bob might not have the PKV routine on his system (for example, if it is a low-end device) or want to pay the performance cost. Alice may also wish to have PKV done, and she might ask her CA to perform this. The CA could have already done PKV as part of the certification process, this way Bob never sees a public key that does not pass PKV. This detects the problem earlier and therefore has much less of a ripple effect. Alice is grateful that her inadvertent key generation error was detected before distributing her certificate that might allow Eve to forge Alice’s signature. Eve is thwarted as she is not able to obtain a certificate containing an invalid public key.

Note that even if there is a legal agreement or a law assigning responsibility in this situation, at least one party will wish that the error was detected beforehand. For example, a Certificate Practice Statement (CPS) or digital signature law might be written to say that a user is required to accept all signatures that verify. It seems clear in this case that Alice (that is, an honest user) may want to be very sure that there were no errors during key generation and so PKV is useful in this scenario as well.

Let us now look at how PKV can solve other potential problems.

#### Pseudo-Encryption

Alice sends a public key to Bob in a certificate. Bob wants to send a private message to Alice, so he obtains Alice’s encryption certificate, verifies that it is not revoked, encrypts his message to Alice using a newly-generated symmetric key and encrypts the symmetric key with Alice’s encryption public key. This is simply the normal PKI encryption scenario to provide message confidentiality.

Alice receives Bob’s encrypted message and tries to decrypt it. Let us say that in this instance, she detects that something is wrong, the decrypted message is garbage. She investigates further and discovers that the public key that Bob

used is simply wrong and does not correspond to her private key. She determines that something went wrong during the output of the public key and a bit was flipped. She examines the public key closer and (with the aid of a cryptographer) determines that the invalid public key does not represent a hard arithmetic problem and therefore is easy to invert (in some cases it might represent a valid public key that does not match her private key, but most of the time the public key will not be valid).

Oops!

Besides possibly needing the services of a cryptographer to determine what happened, this is simply the **wrong** time to detect this error. After a message is encrypted and sent out, it is too late to discover that something is wrong. Bob sent out a message intending it only to be read by Alice. However, it turns out that anyone can read the message as anyone can invert Alice's invalid public key.

One possible reaction from Bob is: "Alice sent me an invalid public key. It was not my fault that Alice sent me an invalid key. Even though I may suffer embarrassment and/or financial loss because my message was revealed, it was not my fault." Notice that while all these statements may be true, they do not attempt to address the problem; rather, they attempt to assign blame.

A more positive and proactive response by Bob is: "Alice may send me an invalid public key by an honest mistake or a dishonest Eve may send me an invalid key deliberately. It is not my fault if someone sends me an invalid key, but it is my responsibility to determine if an encryption key that I wish to use is invalid before I use it to encrypt a message. This is because I may suffer embarrassment and/or financial loss if my message is revealed." This proactive approach attempts to solve potential problems before they become problems by detecting if an error occurred. Therefore, Bob sees the value of doing PKV.

Notice that Alice also wants to detect any errors involved with her encryption public key, after all, she too might be embarrassed and/or suffer financial loss if a message sent to her is revealed. So she may want to do PKV on her own public key, as a safety measure, to detect inadvertent errors during key pair generation. Rather than Alice doing PKV on her key and Bob doing PKV on Alice's key, the CA can do PKV once on Alice's public key during certification, as a service for Alice, Bob and all other clients of the CA.

Now, let us look more closely at Eve and pseudo-encryption using a deliberate invalid key. Eve can choose to deliberately send Bob an invalid public key, one that results in an "encrypted" message from Bob to Eve that is easy to decrypt. However, Eve can also choose to simply reveal any message from Bob to whomsoever she wishes. A neutral observer may reasonably ask, "Where is the attack? And if there is no attack, why would Eve ever do such a thing?"

Let's look closer. If Eve reveals Bob's message to everyone and tries to embarrass Bob, Eve also reveals to Bob that she cannot keep a secret. If Eve reveals Bob's message (intended only for Eve) to Alice, then there is the possibility that Alice will tell Bob who will then figure out that Eve cannot keep a secret. Eve sooner or later gets a reputation for revealing secrets and Bob sooner or later decides that he does not wish to tell Eve any more secrets. Eve will have difficulty repudiating the idea that she revealed the secrets. However, if Bob uses an invalid public key to send encrypted messages to Eve, Eve may be able to repudiate the idea that she told Bob's secrets to anyone, as it may be unclear whether the error was an honest mistake or a deliberate act.

This potential ambiguity with public key encryption can be (unsurprisingly) addressed by using PKV. Alice should refuse to distribute her encryption public key if it does not pass PKV and Bob should refuse to use anyone else's encryption public key if it does not pass PKV. A CA can do PKV on any encryption public key it certifies, this means PKV is only needed to be done once as a service for all its clients. Eve is thwarted in trying to create ambiguity by introduction of an invalid encryption public key as now it is detected.

#### Key Agreement Scenarios

Key agreement is a method to establish a shared symmetric key between communicating parties such that all parties contribute to the value of the symmetric key and no party can control the final value. Elliptic curve and (normal) discrete logarithm key agreement methods typically achieve this by combining Bob's public key with Alice's private key in a key agreement calculation and vice versa. This is different from the situation with a digital signature or RSA encryption, where the public key operation is totally separate from the private key operation.

Here is a question that the reader should be able to answer by intuition by now: "Do you think it might be dangerous to use an invalid public key when doing key agreement?" If you answered, "YES" then you are correct.

At Public Key Solutions '95, Scott Vanstone discussed his discovery of the "small subgroup" attack where Eve inserts an invalid public key into a key agreement protocol between Alice and Bob and may be able to confine the resulting shared secret to a subgroup with a small number of elements. This "small subgroup" is then able to be exhaustively searched by Eve. This type of attack can be very devastating as Alice and Bob may have no idea that something is wrong as they end up sharing a value, it is just not very secret. Unfortunately, PKS '95 had no official proceedings. Fortunately, this "small subgroup" attack was discussed in a Eurocrypt '96 paper (vOW) and discussions of this attack exist in IEEE P1363, ANSI X9.42, ANSI X9.63 and elsewhere.

At Crypto '97, another paper gave an attack that reveals information about Bob's private key if Eve sends Bob an invalid public key which Bob then uses in key agreement, this is known as the Lim-Lee attack (LL). If an adversary is able to obtain enough information about a private key by repeatedly doing Lim-Lee attacks using different invalid public keys, in some cases the adversary may be able to determine the entire value of the private key. The point is that not only can use of an invalid public key weaken the security of the key agreement (and is analogous to weakening the security of a digital signature or RSA encryption), it might also weaken the security of a private key that could be used for future key agreements.

Again, PKV can help. Alice and Bob can refuse to do a key agreement using an invalid public key. This makes sense for long-term key agreement public keys, for example, those in certificates. The CA can do PKV during the certification process and then all of the clients of the CA know that PKV has been done.

However, some key agreement schemes use one-time ephemeral keys. This is usually done to achieve the security property of forward secrecy. (Forward secrecy is the property that even if you somehow lose your long-term private key, your session key is still unknown, this can be especially important when trying to keep data confidential.)

PKV has a performance cost. For ephemeral keys, the cost is about the same as a Diffie-Hellman key agreement calculation, that is, an elliptic curve scalar multiplication or a discrete logarithm modular exponentiation. However, there is another way to address this possible concern. This is to use a sophisticated key agreement method that uses cofactor multiplication such as is found in ANSI X9.63, ISO 15946-3, or IEEE P1363. This ensures that the final shared secret is not confined to a small subgroup. This approach uses the "mitigation" philosophy, instead of using the "safe default" philosophy of simply refusing to use an invalid public key. This "mitigation" philosophy means that although the other party's ephemeral key-agreement public key may be invalid, the key agreement calculation addresses this possibility and mitigates the negative security effects this might have.

#### Proof of Possession (POP)

Note that there are also subtle attacks that may be possible if a user is able to get a public key certified as hers when it really belongs to someone else. To address these attacks, there is often a requirement that a user demonstrate proof of possession (POP) of the corresponding private key when asking a CA to certify a public key. For a digital signature public key, this is often done by simply asking the claimed owner to produce a signature that the CA can verify using the public key. For a key establishment public key, this can be done by requiring the user to interact with the CA using the natural function of the key (either key agreement or encryption) or by using a zero-knowledge proof to demonstrate possession of the associated private key.

POP and PKV can be viewed as complements of each other. POP shows that a user owns the corresponding private key, but not necessarily that the public key is arithmetically valid. PKV shows that the public key is arithmetically valid, but not necessarily that anyone owns the corresponding private key. Doing both POP and PKV provides a high level of security assurance as they act together in a synergy.

#### Alternatives to PKV

It is certainly not required that a user always require assurance of the arithmetic validity of a candidate public key. Rather, he can choose to accept any risk. For example, when dealing with family members or close friends, one can choose simply to trust their public key. When using a system with a limited liability, I may not wish to pay the cost of doing PKV on my public key, as the cost may be too high in relation to any possible benefits. If one has carefully studied the exact expected use of the public key, one can determine the exact amount of potential security loss that may be at risk; if the loss is small enough, perhaps PKV is not worth the cost.

The point is that any discussion of PKV is incomplete without discussing the performance costs of PKV. If PKV is cheap, I may want to do it all the time. If PKV costs something, then I might want to weigh the costs and benefits. If PKV is very expensive, then it may be warranted only in very high security scenarios.

#### PKV Performance Costs

For elliptic curve and (normal) discrete logarithm public keys, recall that there is a public set of domain parameters that may be shared between a group of communicating parties; each user independently generates a key pair that is associated with the set of domain parameters. There are certain arithmetic properties that a set of domain parameters are supposed to have in order to meet the requirements of the relevant cryptographic standard.

This suggests the specification in the relevant standard of a domain parameter validation routine that will validate a candidate set of domain parameters to ensure they conform to the requirements of the standard. As the domain parameters are public information, these arithmetic checks can be straightforward to do. Such a Domain Parameter Validation (DPV) routine has been specified in ANSI X9.62 ECDSA, ANSI X9.42 DL Key Agreement, ANSI X9.63 EC Key Agreement and Encryption, and is scheduled to be specified in ANSI X9.30 DSA-2. DPV is also specified in ISO 15946-1 draft standard on elliptic curve cryptography and discussed in IEEE P1363.

Note that DPV is only needed to be used if some party gives you a new candidate set of domain parameters, perhaps a more common alternative is to select a set of domain parameters from an approved list. For example, NIST has compiled such a list for use by the US Federal government [NIST]. Another list

has been compiled by the Standards for Efficient Cryptography Group [SECG], these include the NIST curves. In these cases, the sanctioning party asserts that the curve is suitable for cryptographic use.

Once one has assurance of using a valid set of domain parameters, testing a candidate elliptic curve or normal discrete logarithm public key consists of testing to ensure that (1) it lies in the correct range and (2) it has the correct order. For elliptic curve systems, this means that a candidate public key should (1) be a point on the elliptic curve specified by the domain parameters and (2) be a point on the prime order subgroup generated by the generating point  $G$ . For normal discrete logarithm systems, this means that the candidate public key should (1) be on the correct interval of integers specified by the domain parameters and (2) be an element of the prime order subgroup generated by the generator  $g$ .

Note that PKV in these cases is an offline process, it only needs to examine the public key and its associated set of domain parameters. Also, the validation is 100%, doing PKV means that the associated private key can logically exist. Because of these facts, the EC/DL PKV decision can be viewed as two-tiered, as follows:

- 1) No PKV.
- 2) Offline, full PKV.

The net is that these PKV tests for EC/DL public key are very straightforward. The range check has very little cost and the order check costs one typical operation (for EC, an EC point multiplication; for DL, a modular exponentiation). As the costs are minimal, the recommendation of the author is as follows.

For a candidate EC/DL public key being certified by a CA:

- 1) If POP is being done, then certainly PKV should also be done. Doing POP indicates an aversion to certain security risks, doing PKV is appropriate as it averts other security risks. Note that if doing both POP and PKV, it seems appropriate to do PKV first.
- 2) It is appropriate to consider doing PKV for all keys, even for times when POP is not being done. Even if the cost of POP is deemed too high (for example, if using an interactive zero-knowledge proof), PKV can be done in a offline manner by examining only the public key. For example, a CA may simply decide to always do PKV for all EC/DL public keys.

For RSA public keys, the situation is more complicated. There are some limited tests that can be done looking only at a candidate RSA public key, these are in the nature of plausibility tests and will detect some inadvertent errors. Note that doing any checks at all reduces the “target area” that an invalid public key must satisfy. Each specific check works in a synergistic fashion with other checks and increases the chances of detecting an inadvertent error and makes the job of an adversary trying to find an invalid public key that will pass the checks more difficult. Even if it may not be mathematically infeasible for an adversary to

create an invalid public key that passes some partial validity checks, it may be uneconomical or not worth the trouble.

However (without getting into the details), it is not known how to show that a candidate RSA modulus is the product of two primes of about the same size in a feasible amount of time by just looking at the modulus, this is because information about the RSA group structure (such as the number of elements in the group) must be kept secret. Only the owner of the private key knows this information.

There is a protocol that Bob Silverman, RSA Labs, has discovered that show this property; this protocol requires the owner of the private key act as an oracle to answer queries from the validating entity (e.g., the CA). It turns out that one step of the protocol leaks a little information about the RSA private key, for this reason it is called a limited-knowledge protocol, rather than a zero-knowledge protocol.

The net of this discussion is that validating an RSA public key is feasible and can be done, at some cost. This means that RSA PKV can be viewed as four tiered, as follows:

- 1) No RSA PKV.
- 2) Offline, partial RSA PKV (plausibility checks).
- 3) Online Zero-knowledge partial RSA PKV.
- 4) Online Limited-knowledge full RSA PKV.

For this reason, the recommendation of this author is as follows.

For a candidate RSA public key being certified by a CA:

- 1) If a zero-knowledge POP protocol is being done, then it is certainly appropriate to do at least the zero-knowledge PKV protocol also. Note that it makes sense to do PKV before POP.
- 2) If high-security is desired, then use the limited-knowledge full RSA PKV.
- 3) It is appropriate for a CA to consider always running the plausibility tests that examine only the RSA public key. This will help detect some inadvertent errors and will also mean that an adversary will need to overcome this hurdle.

#### Standards

ANSI X9.62-1999 ECDSA was the first cryptographic standard to specify a PKV routine. ANSI X9.42 DL Key Agreement is expected to become a standard in 2000 and contains a PKV routine. ANSI X9.30 DSA is being revised to allow longer key lengths and is expected to specify a PKV routine. ANSI X9.63 EC Key Agreement and Encryption specifies a PKV routine. ISO 15946 Elliptic Curve Cryptography specifies a PKV routine. IEEE P1363 mentions PKV and may have a work item on PKV.

The author thinks it is also time to standardize on the specification for an RSA PKV routine, this would affect ANSI X9.31 RSA Signatures and X9.44 RSA Key

Transport. X9.31 is scheduled to be revised in 2000 so this could be done as part of that effort. At the January 2000 ANSI X9.F.1 meeting, it was decided to include RSA PKV in ANSI X9.44. PKV is also scheduled to be discussed in the IEEE P1363 study group as a potential followon standard.

### Summary

Public Key Validation (PKV) is a valuable component in establishing trust in the Public-Key Infrastructure (PKI) because it helps to establish trust in the arithmetic properties of a candidate public key. Use of an invalid public key can result in a loss of security, in some cases the loss can be total. The safe default is to assume that use of a potentially invalid public key will void all intended security. Each user of the PKI is responsible for determining whether they want assurance regarding the validity of a candidate public key or whether they wish to assume the risk of using an potentially invalid public key.

The performance cost for elliptic curve or (normal) discrete logarithm PKV is small and has been standardized, the performance cost for RSA PKV is feasible and needs to be standardized. PKV can be seen as the complement of proof of possession (POP), doing both provides a high level of security assurance. As part of the certification process, a CA can do POP and PKV for the public keys that it certifies, as a service to its clients. Standards have been written and are being written to specify how to do PKV.

### References

[LL] C. H. Lim and P. J. Lee, "A Key-Recovery Attack on Discrete Log-Based Schemes Using a Prime Order Subgroup," Proceedings of Crypto '97, Springer-Verlag.

[NIST] NIST Recommended Elliptic Curves, July, 1999. Available at <http://www.nist.gov/encryption>.

[SECG] Standards for Efficient Cryptography, Part 2. Available at <http://www.secg.org>.

[vOW] Paul van Oorschot and Michael Wiener, "On Diffie-Hellman key agreement with short exponents," Proceedings of Eurocrypt '96, Springer-Verlag.

### Biography

Don B. Johnson is Director of Cryptographic Standards for Certicom, is a member of Certicom Research, and sits on the Advisory Board of the Standards for Efficient Cryptography Group (SECG). He participates in ISO SC27, ANSI X9, IEEE P1363 and other standards bodies. He was the editor of the X9.62 Elliptic Curve Digital Signature Algorithm (ECDSA) standard.