Committing to Obsolescence

trystimuli

tryst@imu.li https://try.st.imu.li

Abstract. Many protocols would benefit from public key rotation, yet find the complications of a public key infrastructure too onerous. If the secret-public key pair can be used for signing, it might be used to sign a rotation certificate designating a replacement key, but then so can any attacker who obtains the secret key.
This paper presents a broadly applicable two-party procedure for generating an arbitrary secret-public key-pair that is committed to a given signing key-pair. The signing key may then sign a self-evident rotation certificate, increasing resilience against compromise of the active secret key, without the complications and expanded threat model of a public key infrastructure.

Keywords: Public Key Cryptography · Key Rotation · Commitments

1 Introduction

While it is established good practice to periodically rotate cryptographic keys, many systems rely on actively used long-term public keys, without providing for the authenticated replacement of those keys. Current key rotation mechanisms have significant drawbacks, which may explain part of this problem.

The simplest method to authenticate a new active public key is to use the current one. The current key-pair can either sign a certificate endorsing the new public key or authenticate a channel that the new public key is transmitted over. For example, OpenSSH host key notification[4] uses that second method. The primary drawback with all of these methods is that anyone who compromises the current secret key can do everything the legitimate holder of that key can do, including authenticating a new key of their choice.

The predominate scheme is more complicated. Public key infrastructures and web of trust systems use a language to describe purposes, and then mark some keys as trusted to validate purposes, including the purpose of trusting other keys to validate purposes. In the Web PKI this has resulted in a large set of essentially globally trusted actors, for whom establishing accountability is both a huge project[6] and fails to prevent incorrect certification.

Narrowing that down to its essential features for key rotation, one can imagine a long term signing key used only for designating an active public key and its own successor. This eliminates the open-ended trust question of PKI, and

reduces the risk of the signing key leaking, but the indirection still requires uses of the active key-pair to come with the certificates establishing its validity.

2 Committed Keys

Committed keys enable combining the features of authenticating rotation with the current key and authenticating rotation with a pre-shared signing key. Key generation is augmented by deriving a subkey based on the originally generated key and a signing key, which can then later be revealed and used to certify the replacement key.

2.1 Definitions

The target cryptosystem has a set of secret keys, S, a set of public keys, P, and a projection from secret keys to public keys $\mathrm{Pub} : S \to P$. Pub must be infeasible to reverse.

There must be another pair of functions, adding in the set Q, $* : P \to Q \to P$ and $\cdot : S \to Q \to S$, with the property that for all $x \in S$ and $y \in Q$, $\mathrm{Pub}(x \cdot y) = \mathrm{Pub}(x) * y$. That is to say, values in Q must be able to be used to derive subkeys in S and P. These operations must take a uniform distribution on Q and an arbitrary distribution on S and P to a uniform distribution on S and P. In typical case S and $\cdot$ are the set and operation of a group, and $*$ is a group action on P that uses S, but such structure is not required.

The signing cryptosystem has sets S_s as secret keys and P_s as public keys, and producing signatures in some set Σ, from messages in $\{0,1\}^*$. $\mathrm{Pub}_s : S_s \to P_s$, $\mathrm{Sign} : S_s \to \{0,1\}^* \to \Sigma$, and $\mathrm{Verify} : P_s \to \{0,1\}^* \to \Sigma \to \{true, false\}$. Any signature scheme will do.

Finally, this scheme requires a random oracle, typically approximated by a cryptographic hash function, from a tuple of the signing and target public keys to Q, $H : P_s \to P \to Q$.

There are two roles in this system, the user and certification agents. In practice both are likely to be under the control of the same user, but the certification agent might be embedded in a hardware security module, or simply a different software module with distinct access control.

2.2 Setup

The certification agent generates a signing key, s_s, and deriving the corresponding verifying key p_s. This signing-verifying key-pair may be reused for multiple target public keys.

$$s_s \xleftarrow{R} S_s \quad p_s := \mathrm{Pub}_s(s_s)$$

2.3 Key Generation

The user agent generates a secret key, $s_c \in S$, and derives its corresponding public key, the committed value.

$$s_c \xleftarrow{R} S \quad p_c := \mathrm{Pub}(s_c)$$

Either the user agent sends p_c to the certification agent or the certification agent sends the verification key p_s to the user agent. The recipient then calculates the hash of the certificate verifying key and the committed value, and, if that's the certification agent, sends that hash to user agent.

$$h := H(p_s, p_c)$$

The user agent then derives the target secret and public keys using the related functions on the generated secret and public keys:

$$s := s_c \cdot h$$
$$p := \mathrm{Pub}(s) = \mathrm{Pub}(s_c \cdot h) = \mathrm{Pub}(s_c) * h = p_c * h$$

The user agent or certification agent stores the committed value p_c, and the user agent proceeds to use s and p.

2.4 Certificate Issuance

To issue a certificate concerning the target public key, the certification agent requires the committed value, p_c. Assuming it does not have it already, the user agent sends p_c to the certification agent who signs a certificate containing it and any other claims about the public key p.

$$\sigma \leftarrow \mathrm{Sign}(s_s, p_c||...)$$

The certificate $(p_s, \sigma, p_c, ...)$ may then be distributed publicly.

2.5 Certificate Verification

Any recipient may then verify the signature on the certificate and that $p = p_c * H(p_s, p_c)$.

$$\mathrm{Verify}(p_s, \sigma, p_c||...) \quad p \stackrel{?}{=} p_c * H(p_s, p_c)$$

If both checks pass, the certificate is valid.

3 Security Properties and Attacks

These certificates have essentially the same security goals as digital signatures, in particular existential unforgability under various attack scenarios.

As a valid signature is required, any compromise that does not include the signing key trivially reduces to preimage search on the hash function or an existential unforgability under chosen message attack on the signature scheme. However, there are two relevant attacks based on compromise of the signing key.

3.1 Secret Key Security with Knowledge of Signing Key and Certificate

Suppose the attacker has all the keys except the target secret keys. For example, the certificate agent during or after issue of the certificate.

Their goal is to recover the target secret key.

Assume that there exists some algorithm, $A : P \to S_s \to S$, that accepts a public commitment key p_c, a signing key s_s, and returns the private key corresponding to the public key $p_c * H(\text{Pub}_s(s_s), p_c)$, namely $s_c \cdot H(\text{Pub}_s(s_s), p_c)$.

Pick an arbitrary public key $p_r : P$ and signing key $s_z : S_s$. Then apply the algorithm $A(p_r, s_z)$, resulting in $s_r \cdot H(\text{Pub}_s(s_z), p_r)$. All that is left to recover s_r, and thus inverting Pub, is to invert $\cdot$. So long as inverting $\cdot$ is easy, A must be nearly as infeasible as inverting Pub.

3.2 Existential Unforgeability with Knowledge of Secret Key and Signing Key

Assume that an attacker has the certificate signing key and the target secret key, but not p_c, the committed value, or h the hash of the committed value and the verifying key. Additionally assume that they can easily invert Pub and Pub_s, say with a sufficiently capable quantum computer.

Their goal is to create any valid certificate.

A valid certificate requires a valid signature with a signing key and that the committed value used in the certificate transformed by the hash of itself and the key that signed the certificate equal the public key. The attacker has a signing key that they know there is a valid commitment key to match, and can obviously generate more signing keys if they want, so producing a signed certificate is no problem. However, for whatever value of p_s they use, they still must synthesize p_c such that $p = p_c * H(p_s, p_c)$.

Assuming H sufficiently approximates a random oracle, each potential committed value of p_c produces an unrelated random value for $H(p_s, p_c)$. To the extent that the $*$ operation translates a uniform distribution on one of its inputs to a uniform distribution on its output, the attacker is left to a brute force search of P and P_s for a valid solution. Note that the brute force attack may target many public keys at once.

4 Parameterizations for Target Sets and Cryptosystems

While the space of certification cryptosystems is open to all signature schemes, the choice of target cryptosystems is limited to those that support the transformation operations. Almost all public key cryptosystems are based on mathematical structures that support these, but some may be formulated in ways that make tracking the secret keys awkward.

4.1 Prime Order Groups & Integers Modulo the Group Order

For cryptosystems that use elements of a prime order group for P (public keys), and integers modulo that prime order for S (secret keys), Q may be S, $\cdot$ may be defined multiplication in S, and $*$ may be scalar multiplication of group elements (in additive notation).

In a prime order group, multiplying a member of that group by a scalar is a one-to-one function, so iterating over the scalars will produce every member of that group. Therefore, if the scalar is chosen from a uniform distribution, no matter how the initial group member is chosen, the resultant group member will be uniformly distributed across the group. Likewise, integers modulo a prime form a finite field, so the revelation of h by the certificate does not provide any information about s, as any scalar may have been multiplied by h to generate s.

This applies to many elliptic curve cryptosystems, as well as Ristretto255[7] and other applications of Decaf[5] to Edwards, Twisted Edwards, and Montgomery Curves. Many pairing based cryptosystems (like BLS signatures) likewise use this same arrangement for secret and public keys.

4.2 Prime Order Subgroups & Integers Modulo the Subgroup Order

Edwards, Twisted Edwards, and Montgomery Curves all have non-prime orders, but are desirable to use for their constant time and fast implementations. While using Decaf to construct a prime order group from the subgroup makes a more generally robust system, this method does allow the subgroup to be used directly.

The user agent will generate p_c as a point in one of the eight large prime order subgroups, and multiply it by the random scalar, resulting in a p that is also in that subgroup. For the attacker, using a point of low order for p_c would result in a p that is also low order, and adding a point of low order would only make it less likely to reach the same subgroup as p.

Ed25519 Ed25519[2] is a popular signature scheme based on such an Edwards Curve, however it defines its secret keys in $\{0,1\}^{256}$ which are then hashed to produce an expanded secret, the secret scalar in integers modulo the subgroup order and a MAC key in $\{0,1\}^{256}$. To target Ed25519, the secret key must be kept in this expanded form, with the scalar and the MAC key already generated. The key transformation acts only on the secret scalar and public key, while the MAC key may be left untouched or have the secret scalar hashed into it.

4.3 Finite Field Exponentiation and Multiplication

For cryptosystems like Finite Field Diffie-Hellman[3], both S and P (and thus Q), are integers modulo a public prime modulus q. $\cdot$ is multiplication in the resulting finite field, while $*$ is exponentiation.

While multiplication in a finite field produces a uniform distribution if either of its inputs in uniform, exponentiation does not. Exponentiation may produce

cycle lengths of any factor of $q-1$. Implementations should ensure that q is of the form $q = 2q' + 1$, which restricts the cycle length to 1, 2, $(q-1)/2$, and $q-1$. In that case, the only short cycle length bases are 1 and $q-1$, which are both unlikely to be generated and easy to test for.

4.4 CRYSTALS-Kyber

Kyber[1] uses vector of ring values as the secret key, and a matrix of ring values multiplied by that vector plus another vector as the public key.

This naturally implies that our S, P and Q would all be a vector of ring values, $\cdot$ would be vector addition, and $*$ would be defined as adding the matrix multiplied by a vector to the public key.

Dilithium does essentially the same thing, but splits the public key into two sections, which is not obviously compatible with applying the group action.

5 Conclusion

Cryptographically certified public key rotation is an important and often overlooked component of any real-world secure system. Existing schemes are deficient in important ways, either introducing additional complexity into the normal use of the key, opening that process to additional attacks, or both. The committed keys scheme leaves the normal use of the keys untouched and limits the opportunity for further attacks by committing the public key to a dedicated verifying key. This scheme is broadly applicable to common public key cryptosystems, as it requires very little underlying mathematic structure.

Further Work This paper only presents one possible formulation of committed keys, and there may be others that are suited to other types of public key systems, or provide other useful security properties.

While the signature scheme can easily be parameterized with threshold signatures, this paper has not explored how to commit to a signature key during setup of a threshold signature.

This also does not analyze the security of committed keys in a broader context of interactions with the rest of a cryptosystem. For key rotation this is unlikely to have much impact, the old keys are likely to be promptly discredited, but there may be cryptosystem-specific considerations that should be examined.

References

1. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: Algorithm Specifications And Supporting Documentation (Janurary 2021), https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf

2. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering **2** (2012). https://doi.org/10.1007/s13389-012-0027-1
3. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976). https://doi.org/10.1109/TIT.1976.1055638
4. djm, dtucker, millert, markus: OpenSSH Protocol Extensions and Deviations, https://cvsweb.openbsd.org/src/usr.bin/ssh/PROTOCOL?annotate=HEAD
5. Hamburg, M.: Decaf: Eliminating cofactors through point compression. In: Gennaro, R., Robshaw, M. (eds.) Advances in Cryptology – CRYPTO 2015. pp. 705–723. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
6. Stark, E., DeBlasio, J., O'Brien, D.: Certificate transparency in google chrome: Past, present, and future. IEEE Security & Privacy **19**(6), 112–118 (2021). https://doi.org/10.1109/MSEC.2021.3103461
7. de Valence, H., Lovecruft, I., Arcieri, T.: The Ristretto Group (2023-09-24), https://ristretto.group/