

## **IEEE P1363.2 / D20.1 (Draft update)**

# **Draft Standard Specifications for Password-based Public Key Cryptographic Techniques**

**This D20.1 draft update includes proposed changes for PKRS-1 Annex discussion resulting from the March 31 teleconference review of draft D20.**

### **Contents**

<b>D20.1 update</b>	<b>(page 2)</b>
<b>D20.1 update, with changes from D20 highlighted</b>	<b>(page 5)</b>

## **D20.1 update**

### **D.5.4.23 Limiting password guesses**

This section discusses some issues for applications that impose a limit for online password guessing. Some additional concerns for limiting password guessing specifically in PKRS-1 are discussed in D.5.5.3.7.

*Good vs. bad guesses.* If a general limit is imposed, but bad guesses are not separately identified, then correct guesses would have to be included in the limit. This would require special action (such as requiring the password to be changed) when the general use limit is exceeded.

*Denial of service.* Any limit must be considered in light of the tradeoff that a lower limit makes denial of service attacks easier, and a higher limit requires more randomness in the password.

*Group passwords.* When a client is used by an (anonymous) member of a group of users that share the same password, any query limit needs to accommodate all users. Such practice may motivate significantly larger guessing limits.

*Editor's Note*—Consider expanding discussion here on how a PKAS where the server does not commit to a specific password value may not discriminate between users, similar to the PKRS-1 issue discussed in D.5.5.3.7.

### **D.5.5.3.4 Application considerations for PKRS-1**

The extent to which a single PKRS-1 Client trusts an interacting PKRS-1 Server and the ways in which such trust is established or verified may vary, and are beyond the scope of PKRS-1. This section describes some relevant application notes.

#### ***Use with multiple servers***

Both [FK00] and [Jab01] describe use of a single PKRS-1-CLIENT with multiple independent PKRS-1-SERVERs, where the individual keys established with each server are combined into a single client *master key*. A variety of techniques might be used to derive the master key from multiple PKRS-1 retrieved keys and other components. The master key could then be verified by the Client, perhaps using additional information stored on one or more of the servers, and then used to derive static secret keys that unlock sensitive user data.

#### ***Need for key confirmation***

In order to prevent disclosure of  $\pi$  to an untrusted party that knows  $u$ , or one that knows  $w_C$  and controls  $w_S$ , the application protocol that invokes PKRS-1-CLIENT must confirm that the permuted password  $z_g$  (or, equivalently, a value derived from  $z_g$ , such as  $Z$  or  $K_i$ ) is correct before the Client reveals information about  $z_g$  to untrusted parties.

However, unlike the key agreement schemes in this standard, PKRS-1 does not include a key confirmation operation. Methods for key confirmation may vary considerably, such as those described in [FK00] and [Jab01], which derive a key from a PKRS-1 established key in combination with other components. See also D.5.5.3.7 for how key confirmation is related to detecting bad guesses and preventing exhaustive guessing and SDHP attacks.

### **D.5.5.3.5 Security properties of *Redp* for PKRS-1**

Each REDP meets a superset of the security requirements of different schemes in this document, as described in D.5.4.n. The PKRS-1 key retrieval scheme requires that, for any two REDP output values  $x$  and  $y$ , it is impractical to compute the discrete log base  $x$  of  $y$ .

#### D.5.5.3.6 SDHP attack on PKRS-1

This section addresses an attack on PKRS-1 based on the Static Diffie-Hellman Problem (SDHP) discussed in [BG04]. PKRS-1 provides a DH oracle to an adversary, which is both a theoretical concern for those concerned with security proofs and has been at least potentially exploited in the SDHP attack of [BG04]. This attack improves the ability for an adversary to learn a static DH secret key  $x$ , given access to an oracle that provides  $g^x$  for submitted values of  $g$ . The significance of this result depends on the domain parameters and on how the scheme is used.

In an extreme case of the SDHP attack that minimizes the adversary's offline effort, the effort is reduced from about  $r^{1/2}$  operations to about  $r^{1/3}$ , where  $r$  is the order of  $g$ . However, this case would require about  $r^{1/3}$  oracle queries, which may be unrealistically large for some applications.

Some distinct ways to ensure that the SDHP attack provides no advantage over other methods of attack are listed here:

- *Use a bigger group.* The SDHP attack requires at least  $r^{1/3}$  operations. In EC and some DL settings where one might otherwise choose  $r \cong S^2$  to resist collision search methods of discrete log attack (where  $S$  is a number of operations presumably infeasible to compute), one could instead choose  $r \cong S^3$  to render the SDHP attack insignificant. Note that SDHP attack is not significant in typical DL “safe prime” settings, where  $r = (q-1)/2$  and index calculus methods predominate.
- *Constrain the factors of  $(r-1)$ .* Make sure that  $(r-1)$  has a small factor  $u$  where  $(r-1)/u$  is prime and  $u$  is less than a small “cutoff value” as discussed in [BG04]. One suggestion is for  $u$  to be 2.
- *Limit oracle queries.* Make sure that the number of oracle queries that the PKRS-1 server provides to an adversary does not exceed the minimum limit required for SDHP attack, where the limit depends on the factorization of  $(r-1)$  and on the cutoff value mentioned above. (See D.5.5.3.7 for discussion of password guessing in PKRS-1.) While the issues of determining a suitable guessing limit may be complex, when the application enforces a limit that is less than that required by the SDHP attack, the attack is prevented.

To optimize the performance and security of a PKRS-1 system in a way that balances the concerns of SDHP and other attacks, one may need to consider the size of the group ( $r$ ), the factors of  $(r-1)$ , and the number of oracle queries that can be obtained from the system, as well as the applicability of security reductions, as discussed in [BG04].

#### D.5.5.3.7 Online guessing attack in PKRS-1

An application may choose to limit use of a PKRS-1 server to prevent online exhaustive password guessing by an attacker. [FK00] discusses such applications and how each server can track the number of access attempts and impose such a limit. Further discussion of discrimination of and error handling for good vs. bad guesses is in [Jab01]. See also D.5.4.23 for general discussion of password guessing limitations in password-based key establishment schemes. These discussions focus on the problem of exhaustive online password guessing, which may need to be balanced against the threat of denial of service attack.

A secondary benefit of limiting bad guesses in PKRS-1 may be to prevent SDHP attack, as discussed in D.5.5.3.6.

However, an application of a PKRS-1 server may not have access to user-specific information. Such an application may deliberately use a common PKRS-1 server static key in conjunction with multiple users'

distinct passwords to retrieve distinct keys for each user. (Such practice may be used to help protect user privacy.) In such an application, careful consideration is needed to address the concerns of how to discriminate between good and bad guesses and of how to impose limits to balance the threats of SDHP attack, exhaustive password guessing, and mass denial of service attack against a population of users.

Although PKRS-1 does not require the server to confirm the client's retrieved key, as mentioned in D.5.5.3.4, a PKRS-1 server application may perform a key confirmation function to distinguish between good and bad guesses.

## **D20.1 update, with changes from D20 highlighted**

### **D.5.4.23 Limiting password guesses**

This section discusses some issues for applications that impose a limit for online password guessing. Some additional concerns for limiting password guessing specifically in PKRS-1 are discussed in D.5.5.3.7.

*Good vs. bad guesses.* If a general limit is imposed, but bad guesses are not separately identified, then correct guesses would have to be included in the limit. This would require special action (such as requiring the password to be changed) when the general use limit is exceeded.

*Denial of service.* Any limit must be considered in light of the tradeoff that a lower limit makes denial of service attacks easier, and a higher limit requires more randomness in the password.

*Group passwords.* ~~One of the features of PKRS-1 is "blinding", which makes it possible for the client to remain~~ When a client is used by an (anonymous) member of a group sharing the same password, a query limit for a group password would need of users that share the same password, any query limit needs to accomodate all users, which ~~users. Such practice may motivate a significantly larger guessing limits.~~

*Editor's Note*—Consider expanding discussion here on how a PKAS where the server does not commit to a specific password value may not discriminate between users, similar to the PKRS-1 issue discussed in D.5.5.3.7.

### **D.5.5.3.4 Application considerations for PKRS-1**

The extent to which a single PKRS-1 Client trusts an interacting PKRS-1 Server and the ways in which such trust is established or verified may vary, and are beyond the scope of PKRS-1. This section describes some relevant application notes.

#### ***Use with multiple servers***

Both [FK00] and [Jab01] describe use of a single PKRS-1-CLIENT with multiple independent PKRS-1-SERVERS, where the individual keys established with each server are combined into a single client *master key*. A variety of techniques might be used to derive the master key from multiple PKRS-1 retrieved keys and other components. The master key could then be verified by the Client, perhaps using additional information stored on one or more of the servers, and then used to derive static secret keys that unlock sensitive user data.

#### ***Need for key confirmation***

In order to prevent disclosure of  $\pi$  to an untrusted party that knows  $u$ , or one that knows  $w_C$  and controls  $w_S$ , the application protocol that invokes PKRS-1-CLIENT must confirm that the permuted password  $z_g$  (or, equivalently, a value derived from  $z_g$ , such as  $Z$  or  $K_i$ ) is correct before the Client reveals information about  $z_g$  to untrusted parties.

However, unlike the key agreement schemes in this standard, PKRS-1 does not include a key confirmation operation. Methods for key confirmation may vary considerably, such as those described in [FK00] and [Jab01], which derive a key from a PKRS-1 established key in combination with other components.

~~The extent to which the Client trusts the Server and the ways in which such trust is established or verified may vary, and are beyond the scope of PKRS-1.~~

See also D.5.5.3.7 for how key confirmation is related to detecting bad guesses and preventing exhaustive guessing and SDHP attacks.

### D.5.5.3.5 Security properties of *Redp* for PKRS-1

Each REDP meets a superset of the security requirements of different schemes in this document, as described in D.5.4.n. The PKRS-1 key retrieval scheme requires that, for any two REDP output values  $x$  and  $y$ , it is impractical to compute the discrete log base  $x$  of  $y$ .

### D.5.5.3.6 SDHP attack on PKRS-1

~~*Editor's Note* – D20: This new section is adapted from mailing list discussion, and may lead to further guidance or change in P1363.2.~~

This section addresses an attack on PKRS-1 based on the Static Diffie-Hellman Problem (SDHP) discussed in [BG04]. PKRS-1 provides a DH oracle to an adversary, which is both a theoretical concern for those concerned with security proofs and has been at least potentially exploited in the SDHP attack of [BG04]. This attack improves the ability for an adversary to learn a static DH secret key  $x$ , given access to an oracle that provides  $g^x$  for submitted values of  $g$ . The significance of this ~~new~~ result depends on the domain parameters and on how the scheme is used.

~~In an extreme case of the SDHP attack (described in [BG04]) that minimizes the adversary's offline effort, the effort is reduced from about  $r^{1/2}$  operations to about  $r^{1/3}$ , where  $r$  is the order of  $g$ . However, this case would require about  $r^{1/3}$  oracle queries, which may be unrealistically large for some applications.~~

~~Some distinct ways to address this attack are: ensure that the SDHP attack provides no advantage over other methods of attack are listed here:~~

- ~~□ Use a bigger group. Like, use 240 bits instead of 160. Or if you're using an oversized group, like a DL 1023 bit  $r=(q-1)/2$ , you're OK as is.~~
- ~~□ The SDHP attack requires at least  $r^{1/3}$  operations. In EC and some DL settings where one might otherwise choose  $r \cong S^2$  to resist collision search methods of discrete log attack (where  $S$  is a number of operations presumably infeasible to compute), one could instead choose  $r \cong S^3$  to render the SDHP attack insignificant. Note that SDHP attack is not significant in typical DL "safe prime" settings, where  $r = (q-1)/2$  and index calculus. Make sure  $(r-1)$  has no factors in the critical range of  $[C, r^{1/2}]$ . At the low end,  $C$  is a small "cutoff" value. At the high end,  $r^{1/2}$  is probably overkill, but easy to remember. When there's no factor in this range, the new attack provides no advantage over earlier methods. One suggestion is for  $(r-1)/2$  to be prime. (Note: This is different from whether one uses DL  $GF(p)$  with cofactor  $k = (p-1)/r = 2$ .)~~
- ~~— *Editor's Note* – Consider adding discussion for how to compute  $C$ ; methods predominate.~~
- ~~— Constrain the factors of  $(r-1)$ . Make sure that  $(r-1)$  has a small factor  $u$  where  $(r-1)/u$  is prime and  $u$  is less than a small "cutoff value" as discussed in [BG04]. One suggestion is for  $u$  to be 2.~~
- ~~□ Limit oracle queries. Make sure that the number of oracle queries that the PKRS-1 server provides to an adversary is limited to less than that does not exceed the minimum limit required for the new attack. This SDHP attack, where the limit depends on the factorization of  $(r-1)$ , and on the cutoff value.~~
- ~~— value mentioned above. (See D.5.5.3.7 for The last solution might be natural for some applications. Although PKRS-1 does not require the server to confirm the client's retrieved key, an PKRS-1 server~~

~~application may perform this function to detect and limit online guessing of the password. discussion of password guessing in PKRS-1.) While the issues of determining a suitable guessing limit may be complex, when the application enforces a limit that is less than that required by the SDHP attack, the attack is prevented.~~

To optimize the performance and security of a PKRS-1 system in a way that balances the concerns of SDHP and other attacks, one may need to consider the size of the group ( $r$ ), the factors of  $(r-1)$ , and the number of oracle queries that can be obtained from the system, as well as the applicability of security reductions, as discussed in [BG04].

#### **D.5.5.3.7 Online guessing attack in PKRS-1**

~~([FK00] discusses applications of PKRS-1 and recommends that guessing limits be imposed, and furtherhandling of good vs. bad guesses is in [Jab01].) If a small limit is enforced by the application,~~  
whichAn application may choose to limit use of a PKRS-1 server to prevent online exhaustive password guessing by an attacker. [FK00] discusses such applications and how each server can track the number of access attempts and impose such a limit. Further discussion of discrimination of and error handling for good vs. bad guesses is in [Jab01]. See also D.5.4.23 for general discussion of password guessing limitations in password-based key establishment schemes. These discussions focus on the problem of exhaustive online password guessing, which may need to be balanced against the threat of denial of service attack.

A secondary benefit of limiting bad guesses in PKRS-1 may be to prevent SDHP attack, as discussed in D.5.5.3.6.

However, an application of a PKRS-1 server may not have access to user-specific information. Such an application may deliberately use a common PKRS-1 server static key in conjunction with multiple users' distinct passwords to retrieve distinct keys for each user. (Such practice may be used to help protect user privacy.) In such an application, careful consideration is needed to address the concerns of how to discriminate between good and bad guesses and of how to impose limits to balance the threats of SDHP attack, exhaustive password guessing, and mass denial of service attack against a population of users.

Although PKRS-1 does not require the server to confirm the client's retrieved key, as mentioned in D.5.5.3.4, a PKRS-1 server application may perform a key confirmation function to distinguish between good and bad guesses.