EC-SPEKE Authentication Mechanism for AllJoyn

Greg Zaverucha (gregz@microsoft.com)

June 23, 2015

This document describes the design of a new authentication method for AllJoyn, called ECDHE_SPEKE. The method uses a password-authenticated key exchange, similar to the SRP mechanism in AllJoyn. Currently, SRP in AllJoyn allows two endpoints to establish a shared secret, provided they share a password. This is useful for bootstrapping security when onboarding new devices to the AllJoyn network. Unlike the ECDHE_PSK mechanism, the pre-shared value in SRP may have low entropy (like a 4-digit PIN printed on the box of a Thing with no display), and still provide strong authentication security. This is a useful feature, but SRP is not the best way to realize it, for the following reasons:

1. SRP is not supported on the thin client. It's also not a good fit for low-power devices, as it's computationally more expensive than elliptic-curve (EC) based primitives.
2. SRP uses a separate crypto stack from the ECDSA and ECDH. To implement SRP requires large integer arithmetic (bignum) and SHA-1, and SRP is the only authentication mechanism using these. The other mechanisms share the EC code, and all use SHA-256. To add the current SRP protocol to the thin client would cause a non-trivial increase in code size.
3. The current SRP implementation needs work, e.g., it lacks protections against side channel attacks. The considerable investment to improve it is better spent implementing the mechanism described here.

One document specifying the SRP protocol used by AllJoyn is IEEE 1363.2 [IEEE]. This document also specifies other password-authenticated key exchange protocols. IEEE 1363.2 has three scheme types, two of which are relevant for AllJoyn.

1. *Balanced* password-authenticated key agreement. Both parties have a password; the scheme succeeds if they know the same password. The output is an ephemeral shared secret.
2. *Augmented* password-authenticated key agreement. One party is in a Server role, and does not have the password. In place of the password, the Server has data derived from the password with a one-way function. Naturally, the password-derived data is subject to offline attacks (just like a password file), but this is unavoidable – the server must be able to determine whether a particular password is valid.

Both settings are currently supported in AllJoyn with SRP (AUTH_SUITE_SRP_KEYX is balanced, and AUTH_SUITE_SRP_LOGON is augmented). The new design only supports the balanced case. In most AllJoyn scenarios, we expect passwords to be used to bootstrap security between two devices, and once a shared secret is established it will be used for long (e.g., months to years) periods of time. Alternatively, after establishing an authenticated channel with a password the peers will exchange certificates that will be used to authenticate subsequent authentications. If a scenario requires long-term use of the password, this design has the small drawback that the password itself must be stored,

rather than password-equivalent verification data. However, in practice the password may be recovered from the verification data by brute force[1].

The new design uses the SPEKE protocol, first described in [Jablon96], with the elliptic curve parameters currently used in AllJoyn (NIST P256), so we use the name "EC-SPEKE". The protocol is called "BPKAS-SPEKE" in IEEE 1363.2 [IEEE] and is defined in Section 9.4 of that document.

SRP and EC-SPEKE vs ECHDE_PSK

For security in the context of AllJoyn, the entropy of the pre-shared keys used with ECHDE_PSK must be 128-bits (or more). This means PSK values must be generated randomly and be relatively long, and notably excludes the use of a user-chosen password or short numeric PIN. This is because an active attacker may attempt to connect using ECDHE_PSK with an honest participant (the session will fail), then perform a brute force attack to recover the PSK from the observed conversation (see [core-list] for details). Once the attacker has the PSK, he may authenticate successfully.

Since the new EC-SPEKE mechanism will be available in both the standard and thin clients, and works with high entropy PSK values, it could replace the ECDHE_PSK mechanism entirely. Having a single mechanism for authentication with pre-shared values will guard against misuse by applications, since they will not have to choose between EC-SPEKE/SRP and PSK.

We recommend deprecating the ECDHE_PSK mechanism in favor of the EC-SPEKE mechanism.

EC-SPEKE Overview

We now give an informal description of EC-SPEKE. It's similar to ECDHE with the main difference being the base used to compute the ephemeral public keys. In ECDHE, Alice chooses an ephemeral secret key a, and computes her ephemeral public key as $A = g^a$. Similarly, Bob computes $B = g^b$. The value g is part of the group parameters (in AllJoyn, g is the base point of the NIST P256 parameter set). The shared secret is computed as B^a by Alice, and A^b by Bob.

In EC-SPEKE, Alice and Bob derive g from the password they share, using a special function $REDP$, which encodes strings as group elements. Let π be the password Alice and Bob share. They both derive $g_\pi = REDP(\pi)$, and proceed to compute $A = (g_\pi)^a$ and $B = (g_\pi)^b$, and continue as in ECDHE. Note that the values revealed on the wire, A and B reveal no information about π, since for any (A, π), there exists an a such that $A = (g_\pi)^a$. Therefore offline brute force attacks to recover the password are not possible. Second, note that if the passwords do not match, the shared secret will be different and the key exchange will fail.

ECDHE_SPEKE Detailed Specification

The proposed name of the new authentication mechanism is ECDHE_SPEKE (for consistency with the other authentication mechanisms ECDHE_NULL, ECHDE_PSK and ECDHE_ECDSA). This is a profile of Section 9.4.1 - 9.4.2 of IEEE 1363.2 for AllJoyn.

Inputs/Parameters

i) Password π. Array of uint8_t

[1] In most augmented schemes the verification data is equivalent to an unsalted hash of the password, from the perspective of a brute-force attacker. For example, in BSPEKE2 from [IEEE], the verification data includes a group element derived deterministically from the password.

ii) Nonces $nonce_C$, $nonce_S$, and GUIDS $GUID_C$, $GUID_S$ associated with this instance of key exchange.
iii) Curve parameters. Let r be the order of the group.
iv) REDP function to encode byte arrays as points on the curve. AllJoyn will use the REDP-2 function from 1363.2, described below.

Key agreement steps

Before these steps occur, the AllJoyn ECHDE key exchange protocol has exchanged nonces and GUIDs. The password π is provided by the application with a callback.

a) Compute a group element $g_\pi = REDP(\pi||nonce_C||nonce_S||GUID_C||GUID_S)$
b) Choose a random ephemeral private key $s \in Z_r$
c) Compute the ephemeral public key $Q = (g_\pi)^s$. Delete g_π.
d) Send Q and receive Q' from the other party.
e) Validate Q' as a group element:
    a. Q' satisfies the curve equation
    b. Q' is not the identity element

    This implies that Q' is in the correct group, since the curve has prime order (cofactor 1). If validation fails, abort.
f) Compute the field element z = x_coordinate($(Q')^s$). Delete s.
g) Convert z to the uin8_t array Z, by taking the bytes of z in big-endian. The length of Z is ceil(bitlength(p)/8), 32 for NIST-P256.

Key Derivation and Confirmation

h) Z is the agreed secret, now derive keys and key confirmation values (verifiers) as specified in AllJoyn for ECDH:
    a. Let PMS = SHA-256(Z) be the premaster secret. (This is done with the function DerivePreMasterSecret).
    b. Then a 48-byte master secret value is computed as

       MS = TLS_PRF(PMS, "master secret") = MS[0]||MS[1]

       where

       MS[0] = HMAC-SHA-256(PMS, "master secret")

       MS[1] = HMAC-SHA-256(PMS, MS[0]||"master secret") (truncated to 16-bytes)
    c. Then the 32-byte verifiers are computed as: TLS_PRF(MS, handshake_digest||"server finished") and TLS_PRF(MS, handshake_digest||"client finished"). The value handshake_digest is a hash of the conversation between both parties. Since the output length of the TLS_PRF is 32 bytes, the function is equivalent to HMAC-SHA-256 keyed with MS. Upon receiving the remote verifier, it is recomputed and compared. If this verification fails, the endpoint aborts.

The EC-SPEKE implementation can re-use the ECDHE code, once the base element is derived from the password. The message flows also match ECDHE, and the PRF/verifier mechanism used by the ECDHE mechanism may be securely re-used with EC-SPEKE for key confirmation. The similarity to ECDHE means that the EC-SPEKE mechanism will fit into the KeyExchanger class. As with SRP a callback to the application will be required to get the password.

Noncompliance with IEEE 1363.2

List of differences with 1363.2 BPKAS-SPEKE:

- REDP input includes both nonces and GUIDs in addition to the password. This can be seen as compliant, by treating the whole (hashed) string as a password.
- AllJoyn key derivation and confirmation is used. Part of key derivation can be seen as complying with 1363.2, but a second key derivation step is done with the TLS PRF. The inputs to key confirmation are different and the functions used do not match. Key confirmation is considered an optional step for EC-SPEKE in 1363.2.

Discussion

- The deviations from 1363.2 likely preserve the security properties of the standardized version, and will make integrating the protocol into AllJoyn simpler and easier.
- Existence of other 1363.2 EC-SPEKE implementations that will be interoperable is unlikely:
    - EC-SPEKE is not widely implemented
    - Supported curve and hash parameters must intersect
    - Choice of REDP function must be same. If REDP1, the implementations must take square roots in the same way to actually interoperate due to a spec bug. If REDP2 is used, the constants must be same.
    - The (optional) key confirmation steps must be done in both implementations, must be done the same way.
    - Key derivation must be the same.

Encoding passwords as elliptic curve points

Implementing an encoding of strings to elliptic curve points is the most difficult part of the implementation (not already in AllJoyn). This is sometimes described as "hashing into the elliptic curve group". We will use the REDP-2 function from 1363.2, which uses REDP-1 from the same standard.

REDP-1

The direct implementation (called REDP-1 in 1363.2) is to compute, $H(\pi||1)$, $H(\pi||2)$, … until finding $H(\pi||i)$ is a valid x-coordinate of the curve. Unfortunately this is hard to protect against timing attacks. A passive attacker observing the response time of a participant gets some information about the number of iterations required, which in turn leaks information about the password. Attackers may create dictionaries of passwords that require a given number of iterations. Combined with other information (e.g., knowledge that the password is a 4-digit PIN), this may reduce the number possible passwords, likely to a small enough set that online attacks are efficient.

Simple countermeasures like always doing a fixed number of iterations are not practical. For k iterations, one in 2^k passwords will require more than k iterations. So k must be large (64 or above) in order to have a high chance of working for all passwords. If the cost of 64 iterations was small this might be an option, but it is too costly. One could also group iterations (e.g., always perform a multiple of eight iterations), however this will still leak information and have high cost.

For these reasons, we do not use REDP-1 on the password directly, only fixed constants, to implement REDP-2, as described below.

A final note on REDP-1 is that separate implementations are unlikely to interoperate reliably, since there is a bug in the spec. When computing a square root mod p, there are two correct answers, and the two protocol endpoints must have a way to choose one (e.g., always choose the smaller value). REDP-1 computes square roots (step h.3), but does not specify a way to choose one of the two roots. In this implementation, which uses the NIST-P256 parameter set, the square root of a field element a is computed (unambiguously, in constant time) as $a^{(p+1)/4}$ (mod p).

REDP-2

1363.2 specifies another function (REDP-2, Section 8.2.18) that uses two auxiliary generators (which become part of the parameters). The function is $R(\pi) = (h_1)(h_2)^{\pi}$, for generators h_1 and h_2. The parameters h_1 and h_2 should be generated in such a way that no exponential relationship is known between them. This will be done by deriving them with REDP-1 from fixed public strings, as follows:

$h_1 =$ REDP-1(`Alljoyn-ECSPEKE-1`)

$h_2 =$ REDP-1(`Alljoyn-ECSPEKE-2`)

The complete REDP function is

Input: Password and additional data: $D = \pi||nonce_C||nonce_S||GUID_C||GUID_S$. The five fields are concatenated together. Note that all fields but the password have fixed length (this encoding is reversible).

Steps

1. Compute the curve points
   $h_1 =$ REDP-1(`Alljoyn-ECSPEKE-1`)
   $h_2 =$ REDP-1(`Alljoyn-ECSPEKE-2`)
   The inputs to REDP-1 are the byte arrays corresponding to the ASCII representations of the two constant strings. Note that h_1 and h_2 will be precomputed and cached.
2. Compute π' = SHA-256(D). Set the rightmost bit of $\pi'$to zero, so that as a little-endian integer, $\pi'$is less than the group order.
3. Output $h_1 h_2^{\pi'}$.

References

[Jablon96] D. Jablon. “Strong Password-Only Authenticated Key Exchange,” Computer Communication Review, ACM SIGCOMM, vol. 26, no. 5, pp. 5–26, October 1996.

[IEEE] *IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques*. IEEE Std 1363.2-2008. IEEE Computer Society, New York, 2009.

[core-list] Allseen-core discussion list. Password and PINs with ECDHE_PSK. Mar 28th – April 9th, 2015. https://lists.allseenalliance.org/pipermail/allseen-core/2015-April/001580.html