draft-harkins-tls-pwd

Dan Harkins
Aruba Networks

- What?
  - Certificate-less ciphersuites, more secure than PSK
  - Instantiates a PAKE protocol called “dragonfly”
    - Authentication using a password
    - Resistance to off-line dictionary attack
  - No, it’s not patented

- What’s wrong with SRP? Nothing, but…
  - Nice to have EC support
    - While SRP can technically support EC it’s TLS ciphersuites don’t.
  - Finite cyclic group is not fixed for each user
    - With TLS-SRP the group cannot change, with TLS-PWD it can
    - Allows generation of keys that are suitable for ciphersuite’s hash and cipher– e.g. AES-GCM-256 w/HMAC-SHA384 then use p384 or p521, or AES-GCM-128 with/HMAC-SHA256 then use p256
  - Flexibility for things like draft-pkix-est
    - If getting an EC cert might be nice to use an EC group
  - Same key exchange used in another protocol for data plane protection (802.11 mesh, smart grid applications)
    - Nice to do the same thing for control plane protection– straight forward way to provide consistent, system-wide security

How it Works (very broadly)

Alice generates Password Element

password

$\downarrow$

Hash-to-element

$\downarrow$

PE = password element

Alice generates 2 random numbers

rnd-a, mask-a $\overset{\$}{\longleftarrow} Z_q$

Alice sends scalar and element to Bob

scalar-a = (rnd-a + mask-a) mod q -->

element-a = $PE^{-\text{mask-a}}$ mod p -->

Bob generates Password Element

password

$\downarrow$

Hash-to-element

$\downarrow$

PE = password element

Bob generates 2 random numbers

rnd-b, mask-b $\overset{\$}{\longleftarrow} Z_q$

Bob sends scalar and element to Alice

< -- scalar-b = (rnd-b + mask-b) mod q

< -- element-b = $PE^{-\text{mask-b}}$ mod p

Alice and Bob generate pre-master secret

$(PE^{\text{scalar-b}} * \text{element-b})^{\text{rnd-a}} \bmod p$ = pre-master-secret = $(PE^{\text{scalar-a}} * \text{element-a})^{\text{rnd-b}} \bmod p$

How it works (changes to TLS)

```
enum { ff_pwd, ec_pwd } KeyExchangeAlgorithms;

struct {
   opaque salt<1..2^8-1>;
   opaque pwd_p<1..2^16-1>;
   opaque pwd_g<1..2^16-1>;
   opaque pwd_q<1..2^16-1>;
   opaque ff_sscalar<1..2^16-1>;
   opaque ff_selement<1..2^16-1>;
} ServerFFPWDParams;

struct {
   opaque salt<1..2^8-1>;
   ECParameters curve_params;
   opaque ec_sscalar<1..2^8-1>;
   ECPoint ec_selement;
} ServerECPWDParams;

struct {
   select (KeyExchangeAlgorithm) {
     case ec_pwd:
       ServerECPWDParams params;
     case ff_pwd:
       ServerFFPWDParams params;
   } ;
} ServerKeyExchange;
```

```
struct {
   opaque ff_cscalar<1..2^16-1>;
   opaque ff_celement<1..2^16-1>;
} ClientFFPWDParams;

struct {
   opaque ec_cscalar<1..2^8-1>;
   ECPoint ec_celement;
} ClientECPWDParams;

struct {
   select (KeyExchangeAlgorithm) {
     case ff_pwd:
       ClientFFPWDParams;
     case ec_pwd:
       ClientECPWDParams;
   }  exchange_keys;
} ClientKeyExchange;
```

- diff v01 v02
  - Fixing issues with side channel attack mitigation
  - Editorial changes: nits, clean-up
- Big question from Taipei: Is it secure?

Secure Against Passive Attack

- CDH problem:
  - given (g^a, g^b, g)
  - produce g^{ab}
- dragonfly algorithm:
  - given $(ra+ma, PE^{-ma}, rb+mb, PE^{-mb}, PE)$
  - produce PE^{ra*rb}
- Reduction:
  - generate random r1, r2
  - Give attacker $(r1, g^a, r2, g^b, g)$ to produce $g^{(r1+a)*(r2+b)}$
  - But $g^{(r1+a)*(r2+b)} / ((g^a)^{r2} * (g^b)^{r1} * g^{r1*r2}) = g^{ab}$!
- Conclusion:
  - Successful attack against dragonfly would solve CDH problem, which is computationally infeasible

Secure Against Dictionary Attack?

- “doesn't seem likely that the protocol can be proven secure”– Jonathan Katz
- Random oracle model
  - assume no key confirmation step in dragonfly, just scalar and element exchange
  - adversary performs MitM, adding 1 to one side’s scalar
  - adversary issues “reveal” query to obtain secrets of both sides
  - off-line dictionary attack is now possible
- This is too contrived to worry about as a practical attack– there is key confirmation and if both sides are compromised then off-line dictionary attack is the least of your problems– but it is a problem with a formal proof of security (at least in Random Oracle model)

- OK, what do I want?
  - Someone to interoperate with!
  - Ask WG to accept document and move it forward as a Proposed Standard

  or, at the very least

  - Stable, published specification
  - Codepoints for pwd ciphersuites

CipherSuite TLS_FFCPWD_WITH_3DES_EDE_CBC_SHA = (TBD, TBD);
CipherSuite TLS_FFCPWD_WITH_AES_128_CBC_SHA = (TBD, TBD);
CipherSuite TLS_ECCPWD_WITH_AES_128_CBC_SHA = (TBD, TBD);
CipherSuite TLS_ECCPWD_WITH_AES_128_GCM_SHA256 = (TBD, TBD);
CipherSuite TLS_ECCPWD_WITH_AES_256_GCM_SHA384 = (TBD, TBD);
CipherSuite TLS_FFCPWD_WITH_AES_128_CCM_SHA = (TBD, TBD);
CipherSuite TLS_ECCPWD_WITH_AES_128_CCM_SHA = (TBD, TBD);
CipherSuite TLS_ECCPWD_WITH_AES_128_CCM_SHA256 = (TBD, TBD);
CipherSuite TLS_ECCPWD_WITH_AES_256_CCM_SHA384 = (TBD, TBD);