USENIX Association

Proceedings of the 10th USENIX Security Symposium

Washington, D.C., USA
August 13–17, 2001

PDM: Password-Derived Moduli

A new password-based strong authentication protocol

Radia Perlman, SunLabs

Charlie Kaufman, Iris Associates

Overview

- Why Strong Password Protocols
- Previous Strong Password Protocols
- Credentials Download
- PDM

Why, in 2001, do we care about passwords?

Humans are incapable of storing high-quality cryptographic keys, and they have unacceptable speed and accuracy when performing cryptographic operations. They are also large, expensive to maintain, difficult to manage, and they pollute the environment. It is astonishing that these devices continue to be manufactured and deployed. But they are sufficiently pervasive that we must design our protocols around their limitations.

From Network Security: Private Communication in a Public World, Kaufman, Perlman, and Speciner

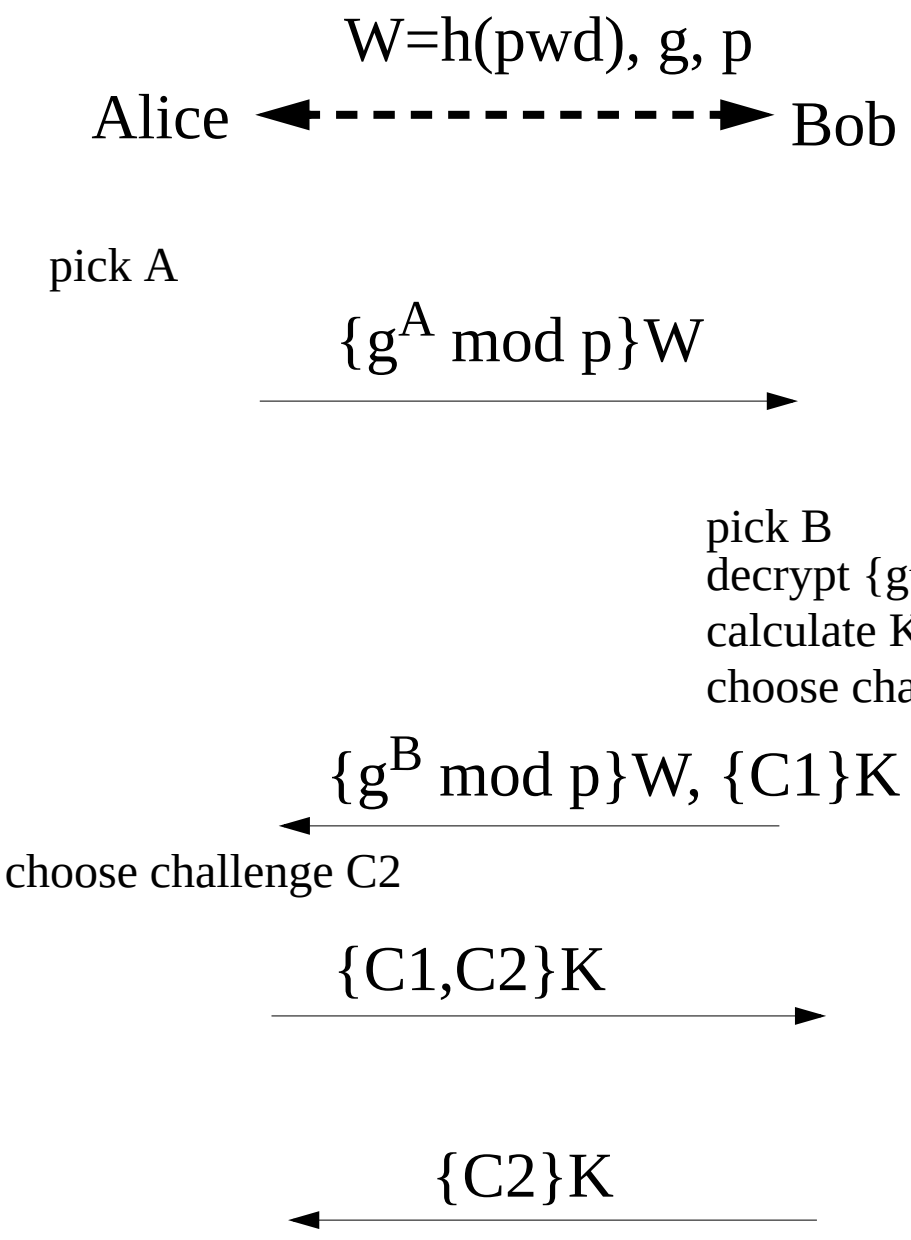
The Vision

- User carries nothing
- Walks up to WS
- WS has correct login software
- WS has no user-specific configuration info
- User types (name,password)
- WS downloads user info
  - private key (encrypted with pwd)
  - trust anchors (signed with private key)
  - browser cookies (encrypted with public key)
- The network is the computer!

Downloading a Private Key from a Public Place

- Private key encrypted with user's password.
- Dictionary attack => can't be world-readable
- ACL won't help (can't prove I'm me until I get the private key)
- SSL
  - Requires trust anchors to be correct.
  - Requires server to have a certificate from a trust anchor
- Using f(pwd) as secret key: dictionary attack, since response is a function of (password, challenge)

EKE: First Such Protocol



Credentials Download

- EKE was designed for mutual authentication
- For credentials download, can be simplified
  - 2 messages
  - stateless Bob
  - have Bob reuse B, to save an exponentiation, and store B and g^B mod p

Two Messages, EKE-based

- Bob stores, for Alice:
  - "Alice"
  - W=h(pwd)
  - Y=private key encrypted with password
  - optionally store B and $g^B \bmod p$

Alice Bob

"Alice", $\{g^A \bmod p\}W$ →

← $g^B \bmod p$, $\{Y\}K$

PDM: Password-Derived Moduli

- Instead of encrypting Diffie-Hellman exchange, use p=f(password)
- Better if p is a “Sophie-Germain” prime: (p-1)/2 is also prime
- To generate Alice’s p:
  - start with filling out chunks of a number of the right size with h(“Alice”, password, constant) in predefined places
  - start searching from there
  - sieve to avoid candidates with small prime factors
  - do primality check

Use 2 as the base

- better if base is a generator
- always use 2, so base isn't dependent on password, and so that you can easily detect if the other side is cheating with an exponent so small 2^A mod p doesn't have to be reduced mod p
- choose p = 11 mod 24
  - 3 mod 8, because then 2 is a generator if p is Sophie-Germain
  - 2 mod 3 so that both p and (p-1)/2 won't be a multiple of 3

PDM for Credentials Download

Alice

Bob

Alice: p, Y

“Alice”, $2^A \bmod p$

$2^B \bmod p$, $\{Y\}(2^{AB} \bmod p)$

Features of PDM

- Unpatented
- Unlikely to infringe on patents of EKE, SPEKE, augmented EKE, or SRP
- Slow (but fast enough) at client
- Potentially much faster at server
  - for traditional D-H, 512-bits within realm of breakability
  - but PDM requires breaking D-H *per password guess*
  - so 512-bit primes probably secure enough
  - smaller prime saves computation at server (half size modulus saves a factor of 6 in computation at server)

Client vs Server Computation

- Client does expensive PDM computation once per session (even if doing mutual authentication with multiple servers)
- Server has to manage a potential large and unpredictable number of clients
- Therefore, server stateless and minimal computation seems like a good idea

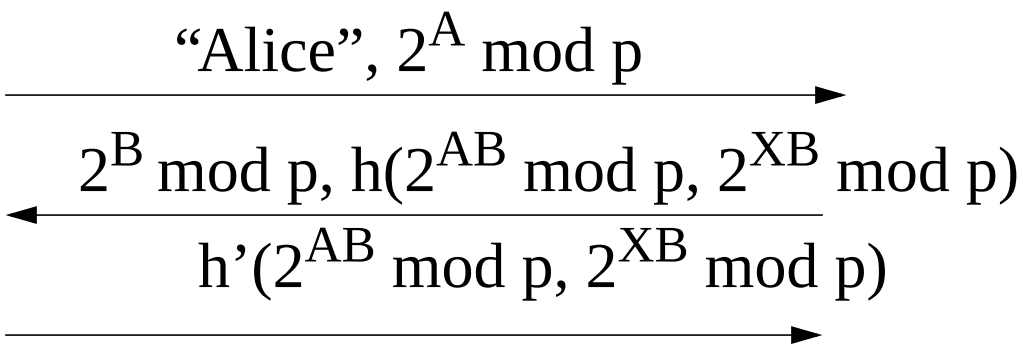
Making PDM Secure

- Avoid Leaking Information
  - Example: naive EKE, if p just a little more than a power of 2, trial decryption of $\{g^A \bmod p\}$ can eliminate about half the passwords. (if decrypted result > p)
  - PDM: choose p from small range (top 65 bits = 111...1110
    - Otherwise, if $2^A > p$ derived from guessed password, can eliminate that pwd
    - Refuse 2^X of form 10000...00000 (not committed to a p since it didn't wrap around)
    - avoid timing attacks (compute p before contacting server)

Avoid Password-Equivalent

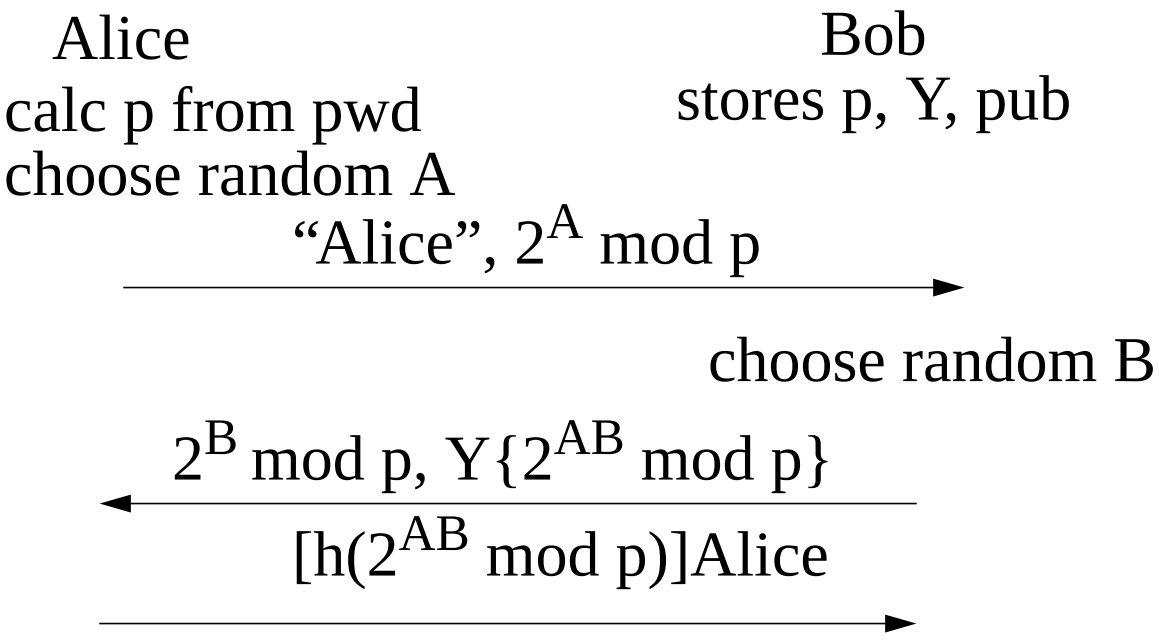
- "Augmented" EKE invented to avoid storing a password-equivalent at the server
- If someone steals the server database, they can't directly impersonate the user (unless they do a successful dictionary attack)
- EKE, SPEKE, and SRP solve this by having the server store $g^{f(pwd)}$ Client needs to know pwd to do its part. PDM approach (X=f(pwd)):

Alice	Bob
calc p and X from pwd	stores p, 2^X mod p
choose random A	choose random B



More Efficient Augmented Protocol

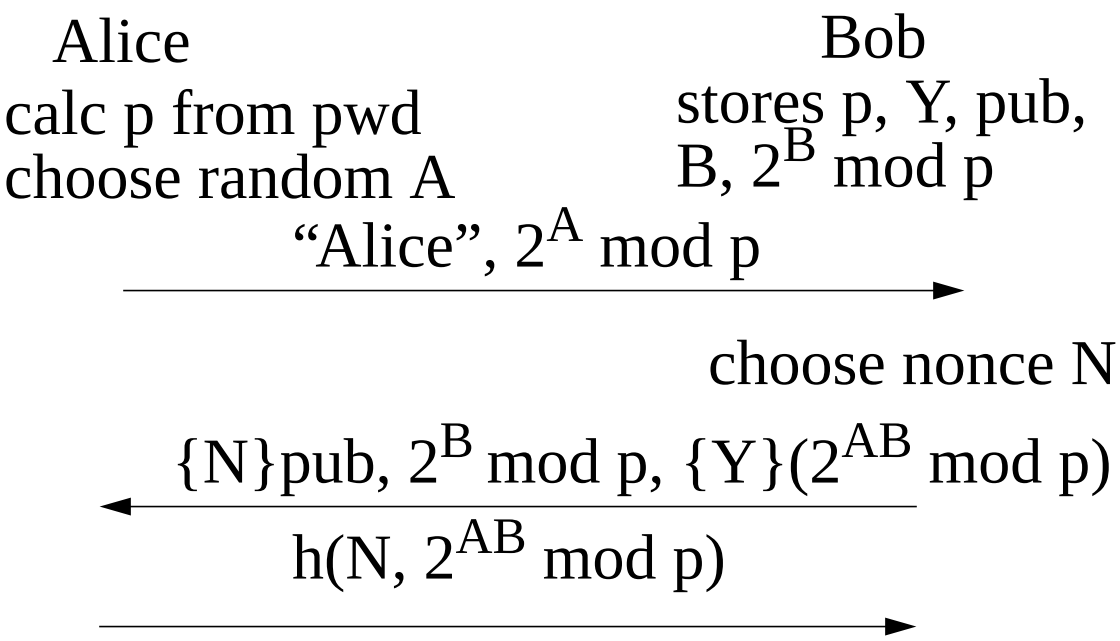
- Do it using RSA instead of extra Diffie-Hellman. Y={priv}pwd



- An RSA verify is cheaper than a Diffie-Hellman exponentiation

Even More Efficient

- Allow Bob to reuse B
- Need a nonce so replay isn't a problem
- Y={priv}pwd



- Even with the same size modulus, this is more efficient for Bob than any previous scheme

Client Performance

- Calculating prime is expensive
- Can cut down time to 1/64 with “hint”
  - First time user chooses pwd, calculate p.
  - Take 6 bits out of p, encode as single char
  - If user furnishes hint, toss out any p’s that don’t have those 6 bits=hint
  - If user forgets hint, takes longer but OK
- Timings on 400 Mhz PC, C code, mean time

prime size	without hint	with hint
512	2.1 sec	.14
768	14.5	.44
1024	41.8	1.0

Summary

- PDM: Password Derived Moduli. A new approach to strong password protocols
- Can be used for 2-message credentials download, or mutual authentication
- RSA-approach to password-equivalent problem new and faster than existing schemes for server
- Client side performance good enough (not intended for wimpy PDAs)
- Can be much faster for server if use smaller moduli