Prof Bill Buchanan OBE Follow
Aug 2, 2018 · 4 min read · · Listen

Living In A GDPR World — Preserving our Information With Zero-Knowledge Proofs (Meet Fiat-Shamir and others)

We give away too much information! Why are we still passing passwords, and why do we still store mod *p*:

Val = G^x (mod *p*)

If we take a simple example with Python:

```
>>> g=92

>>> x=141

>>> p=97

>>> print (g**x) % p

52
```

So even if you know *g* and *p*, it is difficult to determine the value of *x*, and you would have to search for it (and where it will generate an almost infinite number of contenders). So how can you prove to me that I know *x*, without revealing it?

In cryptography, let's say that Alice wants to prove that Bob knows a value (*x*), such that:

g^x (mod p)=Y

where g is a pre-selected value, p is a prime and Y is a result. Both Bob and Alice know these values, and it's difficult to know the value of *x*, as there are many values of *x* that would fit. This is the discrete logarithm problem.

To prove that Bob knows the value of *x*, he creates a random number (*r*) and sends the result of this calculation to Alice:

C=g^r (mod p)

He then sends:

Cipher1=g^(x+r) (mod(p−1))

Alice then calculates:

Cipher2=C.Y (mod p)

```
p= 71  (the prime number)
g= 13
x= 60  (the secret)
r= 8  (the random value)
==============
Y=g**x % p= 32

====Bob sends====
C=g**r % p= 6
Cipher1=g^((x+r)%(p-1)) mod p= 50

====Alice calculates====
Cipher2=C.Y mod P= 50

===============

Well done ... we have proven that you know x
```

Now, let's cheat and use a value of x=61:

```
=== Now we will cheat (x=x+1)====

====Eve sends====
C=g**r % p= 6
Cipher1=g^((x+r)%(p-1)) mod p= 11

====Alice calculates====
Cipher2=C.Y mod P= 50

===============

Not proven. You are a cheat!
```

Non-interactive random oracle access

One of the most widely used methods is the "non-interactive random oracle access" for zero-knowledge proofs. It is defined as the Fiat-Shamir heuristic [1].

In the case we will look at the use of a hash value for the non-interactive random oracle part. Normally Alice would pass a random value to whoever it is that wants to prove her identity, but in this case she will use the hash value to randomise the puzzle and answer.

The stages are:

where we agree on g and x is the secret that Alice proves that she knows. Let's say that g is 13, and x is 11, so g^x is:

```
1792160394037
```

2. Next Alice generates a random number (v) and calculates t, which is:

$t=g^v$

Let's say that the value of v is 8. This gives t of:

```
815730721
```

3. She now computes a hash value (c) created from g, y and t:

c=MD5(g+y+t)

Let's say this gives us 12 (we would normally limit the range of the value produced).

4. Now she computes r of $r=v-c\times t$ to get:

```
-124
```

5. Now she sends out t and r to prove her identity:

```
[815730721, -124]
```

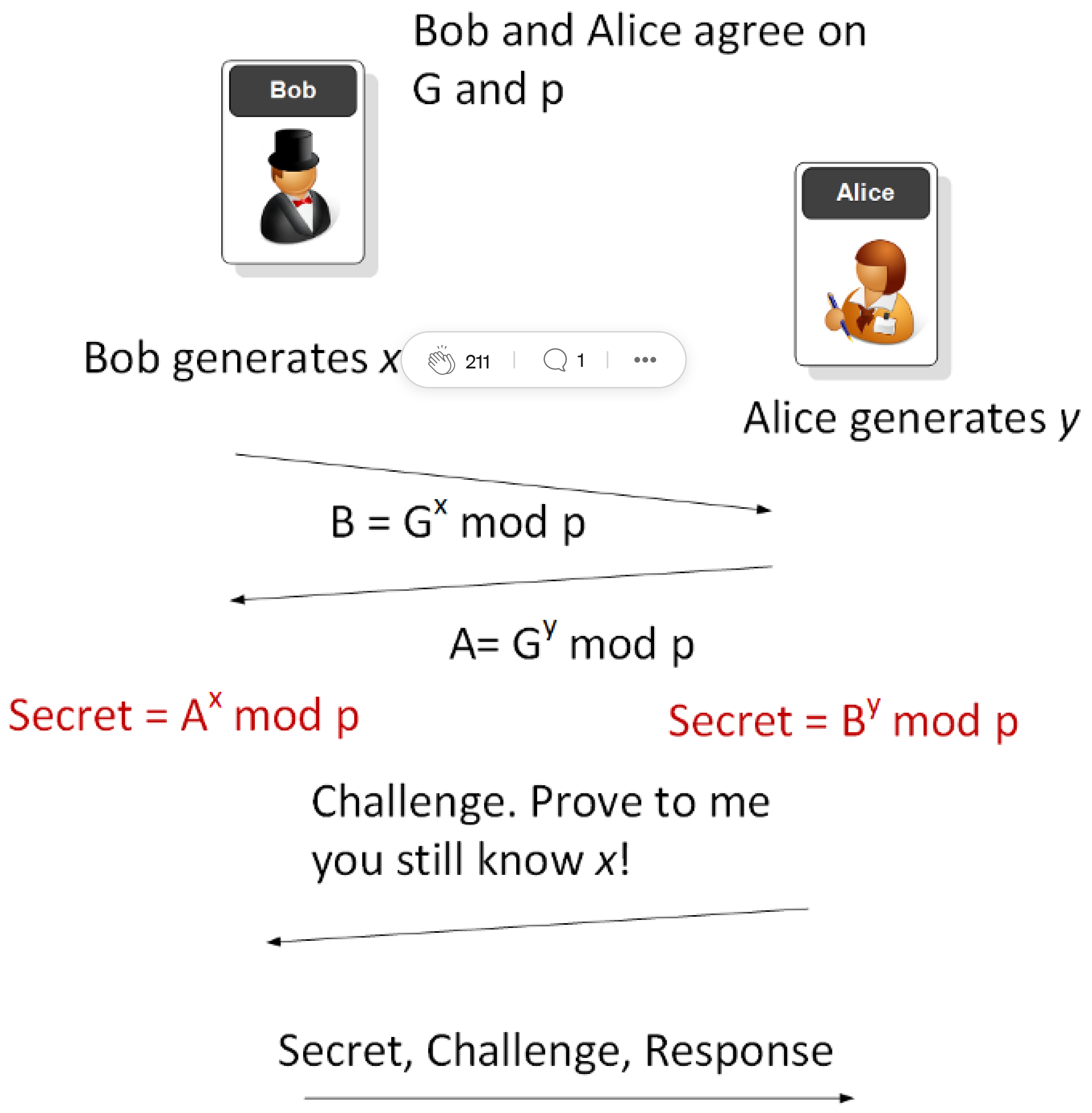
6. Everyone who knows who wants to prove her ID will then compute (where c can be calculated as a hash of g, y and t):

$t=g^r\times y\text{-}c$

In this case the calculation gives 815,730,720, which is the same as the value of t that Alice sent, so

Zero knowledge proof with Diffie-Hellman

Now let's use it in a real-life example. In this case we will use the Diffie-Hellman method to generate a shared key, and then Alice will ask Bob to prove that he still knows the secret value that he initially used.



An example is [here]:

```
x= 16
y= 15

============
a (pub,sec)= 31 16
b (pub,sec)= 62 15
Shared= 87

Now Bob will generate the secret, a challenge and a response
(secret, challenge, response): (87, 322573527190633938482218624508442459321L,
9L)
Alice now checks
Bob has proven he knows x
```

In this method we create a challenge and a response:

```
secret = self.get_shared_secret(remote_pub)

        # Random key in the group Z_q
        randKey = DiffieHellman() # random secret
        commit1 = randKey.public
        commit2 = randKey.get_shared_secret(remote_pub)

        # shift and hash
        concat = str(G) + str(prover_pub) + str(remote_pub) + str(secret) +
str(commit1) + str(commit2)
        h = hashlib.md5()
        h.update(concat.encode("utf-8"))
        challenge = int(h.hexdigest(), 16)
        product = (self.secret * challenge) % phi
        response = (randKey.secret - product) % phi
```

and where the challenge is a hash of G, the prover’s public key, the remote public key, the secret key, a random value and a shared secret random value. PHI is p-1. In this case **secret** will be the shared secret that has been negotiated by Bob and Alice.

Conclusions

We give away so much information, and zero-knowledge proof (ZKP) is an important way for us to limit our leak of data. The basics around the non-interactive random oracle access is still a standard methods that is used in many applications.

[1] Amos Fiat and Adi Shamir: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. CRYPTO 1986: pp. 186–194