DIGITAL SIGNATURES

Typically consists of 3 algorithms:

① The <u>key generation algorithm</u> that generates private key at random. Outputs private key and corresponding public key.

② The <u>signing algorithm</u> that produces signature from a given message and a private key.

③ The <u>signature verifying algorithm</u> that, given a message, public key and a signature, either accepts or rejects the message's claim to authenticity.

BASIC OIL-VINEGAR SIGNATURE SCHEME

The building block of oil-vinegar scheme is the oil-vinegar polynomial. These polynomials are quadratic polynomials in which oil-variables appear linearly. Once we fix values of vinegar variables, the quadratic oil-vinegar polynomial becomes linear in oil-variables. We, then, solve for oil-variables and produce a signature.

k is finite field with q elements

$$\left.\begin{array}{l}\text{oil-variables: } x_1, \ldots, x_o \\ \text{vinegar variables: } \check{x}_1, \ldots, \check{x}_v\end{array}\right\} n = o + v$$

An Oil-Vinegar polynomial is any degree two polynomial $f \in k[x_1, \ldots, x_o, \check{x}_1, \ldots, \check{x}_v]$ of the form

$$f = \sum_{i=1}^{o} \sum_{j=1}^{v} a_{ij} x_i \check{x}_j + \sum_{i=1}^{v} \sum_{j=1}^{v} b_{ij} \check{x}_i \check{x}_j + \sum_{i=1}^{o} c_i x_i + \sum_{j=1}^{v} d_j \check{x}_j + e$$

where $a_{ij}, b_{ij}, c_i, d_j, e \in k$.

Let $F: k^n \longrightarrow k^o$ be a polynomial map of the form

$$F(x_1, \ldots, x_o, \check{x}_1, \ldots, \check{x}_v) = (f_1, \ldots, f_o)$$

where $f_1, \ldots, f_o \in k[x_1, \ldots, x_o, \check{x}_1, \ldots, \check{x}_v]$ are Oil-Vinegar polynomials. Then F is called an Oil-Vinegar map.

OIL-VINEGAR SCHEME

PUBLIC KEY

1) The field k, including its structure.
2) The map $\bar{F} = F \circ L$ such that $\bar{f}_1, \dots, \bar{f}_o \in k[z_1, \dots, z_n]$

PRIVATE KEY

1) The invertible affine transformation $L : k^n \longrightarrow k^n$
2) The Oil-Vinegar map F such that

$$f_1, \dots, f_o \in k[x_1, \dots, x_o, \check{x}_1, \dots, \check{x}_v]$$

SIGNATURE GENERATION

Given document : $(y_1', \dots, y_o') \in k^o$

Choose randomly : $(\check{x}_1', \dots, \check{x}_v') \in k^v$

Calculate : $(x_1', \dots, x_o') = F^{-1}(y_1', \dots, y_n')$, <u>which is equivalent to solving the linear system</u>

$$\underline{F(x_1, \dots, x_o, \check{x}_1', \dots, \check{x}_v') = (y_1', \dots, y_o')}$$

Signature of $(y_1', \dots, y_o')$ is

$$(z_1', \dots, z_n') = L^{-1}(x_1', \dots, x_o', \check{x}_1', \dots, \check{x}_v')$$

SIGNATURE VERIFICATION

Check : $\bar{F}(z_1', \dots, z_n') = (y_1', \dots, y_o')$

BALANCED OIL-VINEGAR ATTACK
($v = o$ so that $n = o + v = 2v = 20$)

IDEA: Structure of associated symmetric matrix allows us to recover another key that is equivalent to original private key.

NEED: $o = v$, given field k has odd characteristic.

GIVEN: Public key polynomials $\bar{f}_1, \dots, \bar{f}_o$

ALGORITHM:

① Write public polynomials in bilinear forms, such that

$$\bar{f}_i = x^T \bar{Q}_i x \quad , \quad \bar{Q}_i \text{ are symmetric matrices.}$$

② For linear combinations of $\bar{Q}_i$ until we get 2 non-singular matrices $\bar{W}_1$ & $\bar{W}_2$.

Calculate $\bar{W}_{12} = \bar{W}_1^{-1} \bar{W}_2$

③ Calculate characteristic polynomial of $\bar{W}_{12}$ ($C(\lambda)$)
Need to make sure that this polynomial has only quadratic factors. If not, repeat steps ①, ②, ③ until we get such $\bar{W}_{12}$.

④ Calculate $C_1(\lambda) = \sqrt{C(\lambda)}$. Evaluate $C_1(\bar{W}_{12})$. This matrix $C_1(\bar{W}_{12})$ has rank o.

⑤ A basis for kernel of $C_1(\bar{W}_{12})$ is o-vectors. Extend this to 20-dimension space to get $(L')^{-1}$. Let $T = (L')^{-1}$

⑥ Calculate $Q_i' = T^T \bar{Q}_i T$

(7) Calculate $f_i' = x^T Q_i' x$, $i = 1, \ldots, o$.

These polynomials are in oil-vinegar format. We use f_i' with T^{-1} to forge signature in the same way a legitimate user would do with original set of oil-vinegar polynomials and corresponding transformation L.

THE PROGRAM

OV_Pub_Key_Gen1 : generates public key & writes onto OV_Pub_Key1

OV_SigVer1 : generates signature using - public key from OV_Pub_Key1 document defined within the file.

Then verifies the signature.

(The result will be "pass" (NOT HARDCODED) because we are verifying the signature against its corresponding document.)

OV_Attack_Part1_1
OV_Attack_Part2_1 } Given a public key, and the ~~signature~~ document, creates another signature (which may or may not be the same as original) but is used to forge the original signature.