Payword

Background

Our main goal is to minimize the number of public-key operations required per payment.

As a rough guide, hash functions are about 100 times faster than RSA signature, and about 10000 times faster than RSA signature generation.

Efficiency Goals

In our schemes the players are brokers(banks and Credit-card companies), users, and vendors.

Try to keep Broker “off-line” as much as possible.

Make purchase/payment efficient, especially for repeated small purchases.

Chain

User computes signs a “commitment” to a new user-specific and vendor-specific chain of paywords $w_1, w_2, \ldots, w_n$.

The user picks the last last payword w_n at random, and computing

$$w_i = h\,(w_{i+1}) \quad \text{for } i = n-1, n-2, \ldots, 0$$

User commits “root” w_0 over to Vender .

Vender redeems commitment, and last payword received, with Broker.

Generalities and Notation

The public keys of the broker B, user U, and vendor V are denoted PK_B, PK_U, and PK_V, respectively;

their secret key are denoted SK_B, SK_U, snd SK_V.

A message M with its digital signature produced by secret key SK is denoted $\{M\}_{SK}$

User-Broker relationship

Certificate

Broker gives User a signed “certificate” C_U good for one month authorizing User to make Payword chains.

C_U = {Broker, User, User’s IP Address, PK_U, expiration-data, limits, etc.}$_{SKB}$

where limits might be a certificate serial number, credit limits to be applied per vendor, information on how to contact the broker, etc.

User-Vendor relationship

Commitment

User commits root w_0 to Vendor by signing commitment message
M_{UV} = {User, Vendor, w_0, C_U, expiration-data}$_{SKu}$

User commits to payword chain for, say, one day.

Note that Broker is not directly involved, and paywords are vendor-specific and user-specific; they are of no value to another vendor.

Verification

The vendor verifies User's signature on M_{UV} and the Broker's signature on C_U (contained within M_{UV}), and check the expiration time.

Note that vendor should cache verified commitments until they expire at the end of the day. Otherwise, User could cheat V for replaying earilier commitments and paywords.

```
bool Verify () {
	if (! VerifyBrokerCertificateSignature())
		return false;
	if (! VerifyUserCommitentSignature())
		return false;
	return true;
}

bool VerifyBrokerCertificateSignature(){
	return this->certificate->verify();
}

bool VerifyUserCommitSignature(){  // Using OpenSSL library
	file_open (PublicKeyFile, this->Certificate->GetUserPublicKey());
	PublicKey = openssl_pkey_get_public('file://', PublicKeyFile);
	bool ok = opensll_verify(this->Commitment, signature, PublicKey);
	file_close();
	return ok == true;
}
```

Payment

A payment P from U to V consists of a payword and its index :

$$P = (w_i, i) .$$

The first payment to Vendor would accompany User's commitment; later payments are just the payword and its index.

Vendor just needs to store one payment from each user : the one with the highest index.

Vendor-Broker relationship

Vendor

Vendor needs to obtain PK_B in an authenticated manner, so he can authenticates certificates signed by Broker.

and sends Broker a redemption message giving, for each of users

(1) the commitment M_{UV} received from User.
(2) the last payment $P = (w_i, i)$ received from User.

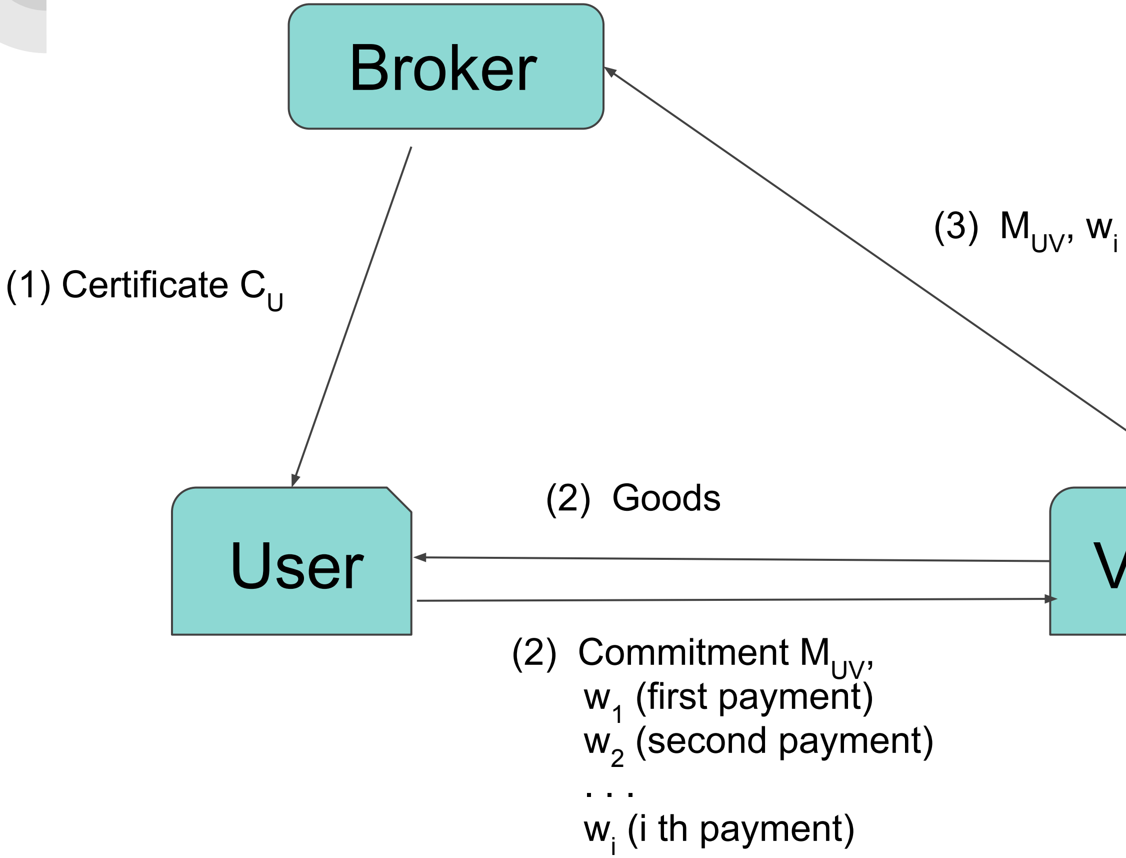
Broker

The broker then needs to

(1) verify user's signatures (since he can recognize his own certification), including checking of datas, etc.

(2) verify each payment (w_i, i) (this requires i hash function applications)

```
Redeem (Commit, UserIdentity, VendorIdentity){
 int amount = 0, TotalPrice = 0;
TotalPrice += (Commit->GetHashChainLength() - 1) * Commit->GetPrice();
        if (! Commit->verify())
                return false;
        for (int i = 0; i < PaywordLength; i++)
             payword = Hash(payword);
        if (payword == Commit->GetFirstPayword())
             amount += Commit->GetPrice() * PaywordLength;
             transfer (User, Vendor, amount);
}
```


Broker
(1) Certificate C_U
(3) M_{UV}, w_i
(2) Goods
User
Vendor
(2) Commitment M_{UV},
w_1 (first payment)
w_2 (second payment)
. . .
w_i (i th payment)

Costs

One signature by Broker / User / month (C_U)

One signature by User / Verdor / day (M_{UV})

Two verifications by Vendor / User / day (C_U and M_{UV})

One verification by Broker / User / Vendor / day (for M_{UV})

One hash function computation by each of User, Vendor, and Broker for each payment.

Variations and Extensions

Can pay for 5 item by revealing w_{10} after w_5.

The value of each payword might be fixed at one cent, or might be specified in C_U or M_{UV}.

In a variation, M_{UV} might authenticate several chains, whose paywords have different values.

Reference

https://people.csail.mit.edu/rivest/pubs/RS96a.slides.pdf

https://people.csail.mit.edu/rivest/pubs/RS96a.prepub.pdf

https://github.com/cretueusebiu/payword/tree/master/app/Payword