Lucre: Anonymous Electronic Tokens v1.8

Ben Laurie
ben@algroup.co.uk

June 3, 2003

1 Introduction

This is a revised version of the theory of blinded coins that may not violate Chaum's patent[1], based on the original work by David Wagner, and conversations with Ian Goldberg, David Molnar, Paul Barreto and various Anonymouses.

Note that this now includes variants that probably do violate the patent, but are of sufficient academic interest to be worthy of inclusion.

2 Coins

2.1 Creating the Mint

The mint chooses a prime, p, with $(p-1)/2$ also prime, a generator, g, s.t.

$$g^2 \neq 1 \pmod{p} \tag{1}$$

and

$$g^{(p-1)/2} = 1 \pmod{p} \tag{2}$$

(see 9.1) and a random number, k,

$$k \in [0, (p-1)/2) \tag{3}$$

Let G be the group generated by g.

The mint publishes

$$(g, p, g^k \pmod{p}) \tag{4}$$

2.2 Withdrawing a Coin

To withdraw a coin Alice picks a random x, the coin ID, from a sufficiently large set that two equal values are unlikely to ever be generated[2], and calculates,

$$y = \mathrm{oneway}(x) \tag{5}$$

(see 9.2). y should be in G; check that

$$1 < y < p - 1 \tag{6}$$

We should avoid the trivial values 1 and -1, because their signatures are independent of k. Note that many one-way coin functions (including the one

[1] At least, that's what people think. Take legal advice before using this stuff!

[2] Remember that if the size of the set of all possible coins is C, the probability of two being the same is .5 after around $\sqrt{C}$ coins have been generated.

presented here) provably never produce 1 or -1, but we include this condition for completeness.

$$y^{(p-1)/2} = 1 \pmod p \tag{7}$$

If it is not, a new coin should be chosen. Note that great care must be take if you want to choose a one-way function that guarantees membership of G - certainly one attempt (see 9.4) led to disaster.

Alice chooses a random blinding factor $b \in [0, (p-1)/2)$ and sends yg^b (the coin request) to the mint. The mint debits Alice's account and returns the blinded signature,

$$m = (yg^b)^k \pmod p \tag{8}$$

Alice unblinds m, calculating the signature,

$$z = m(g^k)^{-b} = (yg^b)^k g^{-kb} = y^k g^{bk} g^{-kb} = y^k \pmod p \tag{9}$$

The coin is then

$$c = (x, z) \tag{10}$$

2.3 Spending a Coin

To spend a coin, Alice simply gives the coin, c, to Bob. Bob then sends it to the mint to be checked. The mint first ensures that x has not already been spent, and that $\mathrm{oneway}(x)$ is in G and is not 1 or -1, then checks that z is a signature for x (i.e. $z = \mathrm{oneway}(x)^k \pmod p$). The mint then records x as spent and credits Bob's account.

3 Attack[6]

Unfortunately an attack on the anonymity of this protocol is possible. The mint can mark a coin in a way that only it can detect, by signing it with k' instead of k. Then the unblinded "signature" is

$$z = (yg^b)^{k'} g^{-bk} = y^{k'} g^{b(k'-k)} \pmod p \tag{11}$$

When Bob submits c to the mint, then the mint calculates

$$y(zy^{-k'})^{1/(k'-k)} = y(g^{b(k'-k)})^{1/(k'-k)} = yg^b \pmod p \tag{12}$$

The mint can then simply look up who sent yg^b to it and thus learn Alice's identity.

4 Type I Defence

One defence against this attack is to make the mint prove that it has signed with k and not some other number. Since the mint must not reveal k, this proof must be a zero-knowledge proof. Two possible zero-knowledge proofs are known to me.

4.1 Variation 1[1]

Given a coin request, yg^b, the mint chooses a random number r s.t.

$$r \in [1, p-1) \tag{13}$$

s.t. r is invertible modulo $p-1$ (i.e. $gcd(r, p-1) = 1$) and calculates

$$t = k/r \pmod{p-1} \tag{14}$$

($p-1$ rather than p because r and t will be used as exponents modulo p). The mint then sends Alice

$$Q = (yg^b)^r \pmod{p} \tag{15}$$

and

$$A = g^r \pmod{p} \tag{16}$$

Alice then randomly demands one of r or t.

If Alice chose r, she verifies that

$$Q = (yg^b)^r \pmod{p} \tag{17}$$

and

$$A = g^r \pmod{p} \tag{18}$$

If Alice chose t, she verifies that

$$A^t = g^{rt} = g^k \pmod{p} \tag{19}$$

and

$$Q^t = (yg^b)^{rt} = (yg^b)^k = z \pmod{p} \tag{20}$$

Note that a mint that wants to cheat has a .5 chance of getting away with it each time (by guessing whether the challenger will choose r or t and lying about Q and A appropriately). Naturally, it is increasingly unlikely to get away with this with each repetition. A suspicious challenger could always repeat the protocol until the probability of cheating is low enough to make them happy.

4.2 Variation 2[2]

The mint chooses a random value r and sends Alice

$$u = g^r \pmod p \tag{21}$$

and

$$v = (yg^b)^r \pmod p \tag{22}$$

Alice responds with a challenge d. The mint answers with

$$w = dk + r \pmod{(p-1)/2} \tag{23}$$

Alice verifies that

$$g^w = g^{dk+r} = (g^k)^d u \pmod p \tag{24}$$

and

$$(yg^b)^w = (yg^b)^{dk+r} = ((yg^b)^k)^d v = (yg^b)^d v \pmod p \tag{25}$$

4.3 Non-interactive variant

It is suggested that choosing

$$d = hash(u, v) \tag{26}$$

would allow the second variation to be used non-interactively. The mint sends (d, w) along with the coin, Alice calculates

$$g^w (g^k)^{-d} = u \pmod p \tag{27}$$

and

$$(yg^b)^w m^{-d} = v \pmod p \tag{28}$$

and verifies that $d = hash(u, v)$.

I'm not entirely convinced that it isn't possible to search for (or even calculate) a set of values that makes this appear to work whilst still signing with k'.

5 Type II Defence[6]

Another defence is to combine two blinding methods, using two indepenent random blinding factors. With this method, the coin-withdrawal protocol changes as follows.

To withdraw a coin Alice picks a random x, the coin ID, from a sufficiently large set that two equal values are unlikely to ever be generated, and calculates,

$$y = \text{oneway}(x) \tag{29}$$

(see 9.2). y should be in G; check that

$$y^{(p-1)/2} = 1 \pmod{p} \tag{30}$$

Alice chooses random blinding factors $b_y, b_g \in [1, p-1)$, ensuring that b_y is invertible modulo $p-1$ (i.e. $gcd(y, p-1) = 1$) and sends $y^{b_y} g^{b_g}$ (the coin request) to the mint. The mint debits Alice's account and returns the blinded signature,

$$m = (y^{b_y} g^{b_g})^k \pmod{p} \tag{31}$$

Alice unblinds m, calculating the signature,

$$\begin{aligned} z &= (m.(g^k)^{-b_g})^{1/b_y} && (32) \\ &= ((y^{b_y} g^{b_g})^k g^{-kb_g})^{1/b_y} && (33) \\ &= (y^{kb_y} g^{kb_g} g^{-kb_g})^{1/b_y} && (34) \\ &= (y^{kb_y})^{1/b_y} && (35) \\ &= y^k \pmod{p} && (36) \end{aligned}$$

not forgetting that $1/b_y$ must be calculated modulo $p-1$, since it is used as an exponent.

Now z is in the same form as in the original scheme and we can proceed as normal.

5.1 Failed Attack

If the mint attempts to mark the coin, as before, then let's see what happens. The blinded signature is

$$m = (y^{b_y} g^{b_g})^{k'} \pmod{p} \tag{37}$$

unblinding, Alice gets

$$
\begin{aligned}
z &= (m.(g^k)^{-b_g})^{1/b_y} && (38)\\
&= ((y^{b_y} g^{b_g})^{k'} g^{-kb_g})^{1/b_y} && (39)\\
&= (y^{k'b_y} g^{k'b_g} g^{-kb_g})^{1/b_y} && (40)\\
&= (y^{k'b_y} g^{(k'-k)b_g})^{1/b_y} && (41)\\
&= y^{k'} g^{(k'-k)b_g/b_y} \pmod p && (42)
\end{aligned}
$$

Because this result entangles both the unknown (to the mint) value y and the, also unknown, value g^{b_g/b_y}, the mint cannot even verify that this is a correct signature, let alone figure out who gave it the blinded coin in the first place.

6 Type III Defence

It has recently been pointed out that the Decisional Diffie-Hellman (DDH) problem has to be separated from the Diffie-Hellman problem (also known as the Computational Diffie-Hellman problem) (DH). In particular there are groups where DDH is easy even though DH is hard. What this actually means in practice is that although given g, g^a and g^b we can't find an h s.t. $h = g^{ab}$, we can, given g, g^a, g^b and g^c, determine whether $ab = c$ (all modulo p, of course).[3]

Given such a group, it is possible to verify that the signature is correct in single blinding without a zero knowledge proof. This works like this: once the coin x has been signed, we have $y = oneway(x)$, a number z that we hope is y^k (but let us signify our doubt for now by calling it $y^{k'}$), g and g^k.[4]

We can check the correctness of z using the easiness of DDH as follows: since g is a generator for the group, there must exist a t s.t. $y = g^t$. Then we have g, g^t (which is y), g^k and $g^{tk'}$ (which is $y^{k'} = z$). We can use easy DDH to check whether $tk = tk'$. If it is, then, of course, $k' = k$ and the signature is genuine.

Of course, the groups we are talking about here are not Z_p^*. In fact, the groups discovered so far with this property are carefully constructed elliptic curves.

Note that there may well be an argument that the weakness of DDH in this group means that the blind signature constitutes a verifiable signature as covered by Chaum's patent and hence would no longer sidestep the patent.

7 Cost and Value

Although there are those that hold that a coin should have a value similar to its cost of production, this is clearly insane, at least when the coin is to be used as money[3].

In general, the cost of production should be considerably less than the value of the coin. So, it is worth calculating the cost of producing Lucre coins.

[3] A clear example where it is not insane is Adam Back's hashcash used as an anti-spam measure - in that case, the whole point is that the coin is expensive to produce.

Assuming that the coins are relatively low value, then a 512 bit signing key should be sufficient. The cost of producing a coin is really the cost of signing it twice (once blinded when withdrawn, and once ublinded when deposited). Implemented in Java on a 300 MHz Pentium[4] we can achieve 25 signs per second. A server in the Bunker (http://www.thebunker.net/) costs £250 per month.

That's £8 per day. 30p per hour, .5p per minute, .001p per second, .0004p per sign.

So, values of .01p per coin are easily achievable.

Incidentally, signing with a 1024-bit key takes around 6 times as long, so values of .1p with 1024-bit security are also achievable.

8 Choosing Parameters

When creating a mint, there are a number of parameters to choose, some of which must satisfy certain properties. Also, some may be well-known and some have to be secret. This section discusses these requirements.

8.1 The Prime, p

The prime, as noted in 2.1, needs to be a Sophie Germain prime[5]. There's no particular requirement for secrecy, so an existing prime that has been proven prime should be preferred to a probabilistically generated one.

The other factor in choosing the prime is a cost/security tradeoff. If the prime is too small, there is a danger that an attacker can retrieve it from coin signatures by brute force. If it is too large, then the cost of signing coins can become prohibitive. A detailed analysis of this tradeoff may be the subject of another paper, but for now (i.e. in February 2003), I would say that a 512 bit prime should only be used for very low value coins. 1024 or 2048 bits should be safe for most applications.

Where can such primes be found? RFC 2412, Appendix E, contains (allegedly) proven 768, 1024 and 1536 bit primes, though certificates are not included. Nor is it entirely clear that $(p-1)/2$ have also been proven prime.

The Internet Draft draft-ietf-ipsec-ike-modp-groups-05.txt adds 2048, 3072, 4096, 6144 and 8192 bit primes - but with no statement of proof (and obviously no certificates). Sinced I-Ds are ephemeral, I include the primes here.

$$2^{1536} - 2^{1472} - 1 + 2^{64}(2^{1406}\pi + 741804) \tag{43}$$

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74
020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437
4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D C2007CB8 A163BF05
```

[4] Surely nothing can be slower that this?

[5] Strictly, a Sophie Germain prime is the *smaller* of the two, and we want the larger.

```
98DA4836 1C55D39A 69163FA8 FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB
9ED52907 7096966D 670C354E 4ABC9804 F1746C08 CA237327 FFFFFFFF FFFFFFFF
```

$$2^{2048} - 2^{1984} - 1 + 2^{64}(2^{1918}\pi + 124476) \tag{44}$$

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74
020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437
4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D C2007CB8 A163BF05
98DA4836 1C55D39A 69163FA8 FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB
9ED52907 7096966D 670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AACAA68 FFFFFFFF FFFFFFFF
```

$$2^{3072} - 2^{3008} - 1 + 2^{64}(2^{2942}\pi + 1690314) \tag{45}$$

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74
020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437
4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D C2007CB8 A163BF05
98DA4836 1C55D39A 69163FA8 FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB
9ED52907 7096966D 670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD33170D 04507A33
A85521AB DF1CBA64 ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7
```

$$2^{4096} - 2^{4032} - 1 + 2^{64}(2^{3966}\pi + 240904) \tag{46}$$

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74
020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437
4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D C2007CB8 A163BF05
98DA4836 1C55D39A 69163FA8 FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB
9ED52907 7096966D 670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD33170D 04507A33
A85521AB DF1CBA64 ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7
ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B F12FFA06 D98A0864
D8760273 3EC86A64 521F2B18 177B200C BBE11757 7A615D6C 770988C0 BAD946E2
08E24FA0 74E5AB31 43DB5BFC E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7
88719A10 BDBA5B26 99C32718 6AF4E23C 1A946834 B6150BDA 2583E9CA 2AD44CE8
DBBBC2DB 04DE8EF9 2E8EFC14 1FBECAA6 287C5947 4E6BC05D 99B2964F A090C3A2
233BA186 515BE7ED 1F612970 CEE2D7AF B81BDD76 2170481C D0069127 D5B05AA9
93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34063199 FFFFFFFF FFFFFFFF
```

$$2^{6144} - 2^{6080} - 1 + 2^{64}(2^{6014}\pi + 929484) \tag{47}$$

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74
020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437
4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED
EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D C2007CB8 A163BF05
98DA4836 1C55D39A 69163FA8 FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB
9ED52907 7096966D 670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B
E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD33170D 04507A33
A85521AB DF1CBA64 ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7
ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B F12FFA06 D98A0864
D8760273 3EC86A64 521F2B18 177B200C BBE11757 7A615D6C 770988C0 BAD946E2
08E24FA0 74E5AB31 43DB5BFC E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7
88719A10 BDBA5B26 99C32718 6AF4E23C 1A946834 B6150BDA 2583E9CA 2AD44CE8
DBBBC2DB 04DE8EF9 2E8EFC14 1FBECAA6 287C5947 4E6BC05D 99B2964F A090C3A2
233BA186 515BE7ED 1F612970 CEE2D7AF B81BDD76 2170481C D0069127 D5B05AA9
93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34028492 36C3FAB4 D27C7026
C1D4DCB2 602646DE C9751E76 3DBA37BD F8FF9406 AD9E530E E5DB382F 413001AE
B06A53ED 9027D831 179727B0 865A8918 DA3EDBEB CF9B14ED 44CE6CBA CED4BB1B
DB7F1447 E6CC254B 33205151 2BD7AF42 6FB8F401 378CD2BF 5983CA01 C64B92EC
F032EA15 D1721D03 F482D7CE 6E74FEF6 D55E702F 46980C82 B5A84031 900B1C9E
59E7C97F BEC7E8F3 23A97A7E 36CC88BE 0F1D45B7 FF585AC5 4BD407B2 2B4154AA
CC8F6D7E BF48E1D8 14CC5ED2 0F8037E0 A79715EE F29BE328 06A1D58B B7C5DA76
F550AA3D 8A1FBFF0 EB19CCB1 A313D55C DA56C9EC 2EF29632 387FE8D7 6E3C0468
043E8F66 3F4860EE 12BF2D5B 0B7474D6 E694F91E 6DCC4024 FFFFFFFF FFFFFFFF
```

Certificates for all of these can be found at ftp://ftp.ssh.com/pub/ietf/ecpp-certificates, however, I am not aware of source for code that allows the certificates to be checked.

8.2 The Generator, g

Equations 1 and 2 constrain the generator. Luckily, there's a trivial choice that satisfies them always: 4.

8.2.1 Proof

$$4^2 = 16 = 1 \pmod{15} \tag{48}$$

and 15 isn't prime.

$$n^{p-1} = 1 \pmod{p} \tag{49}$$

by Euler's theorem, so

$$2^{p-1} = 4^{(p-1)/2} = 1 \pmod{p} \tag{50}$$

which satisfies 2 for all primes.

8.3 Coin Size

The coin needs to be large enough that collisions are extremely unlikely. The only other constraint is that the total size needs to be the same as the key size, so it should not be so large as to reduce the size of the stuff added by the one-way coin function (see 9.2) to a level were coin signatures might be forged. Generally 128 bits is considered sufficient to avoid collisions in most real-world situations. Since we've stipulated a minimum key size of 512 bits, there's no reason to fiddle with this, so 128 bits for the random part is sufficient.

Note that even though the client generates the coin, there's no reason to defend against a client generating the same coin twice (or the same coin as someone else), since that would cause their coin to appear to be a doublespend and would cost them money.

9 Theory

9.1 Subgroup Order

(2) ensures that the order of the subgroup generated by g is $(p-1)/2$.

9.1.1 Leakage

This avoids leakage of information about k which can occur if g generates the whole of Z_p^*, because

$$(g^k)^{(p-1)/2} \begin{cases} = 1 & \text{if } k \text{ is even} \\ \neq 1 & \text{if } k \text{ is odd} \end{cases} \tag{51}$$

Proof

If k is even, then there exists an n s.t. $k = 2n$.

$$(g^{2n})^{(p-1)/2} = (g^n)^{p-1} \tag{52}$$

Since

$$gcd(g^n, p) = 1 \tag{53}$$

then, by Euler's theorem,

$$(g^n)^{p-1} = 1 \pmod{p} \tag{54}$$

If k is odd, then there exists an n s.t. $k = 2n + 1$.

$$(g^{2n+1})^{(p-1)/2} = (g^n)^{p-1} g^{(p-1)/2} \tag{55}$$

$$(g^n)^{p-1} = 1 \pmod{p} \tag{56}$$

(see (54)) and

$$g^{(p-1)/2} \neq 1 \pmod p \tag{57}$$

because the order of g is $p-1$, so no $y < p-1$ can give $g^y = 1 \pmod p$. So

$$(g^n)^{p-1} g^{(p-1)/2} = 1 \cdot x \pmod p, x \neq 1 \tag{58}$$

9.1.2 Subgroup Order Revisited

It has been pointed out that using a g that generates the whole group Z_p^* and choosing k odd also fixes both the above problems, and makes some parts of the protocol cheaper (because you can avoid the exponentiation in the one-way function). This seems to me to be somehow less satisfying, but I can't see anything actively wrong with it.

9.2 One-way Coin Function

The purpose of the one way function is to prevent Alice from cheating the mint by producing variants on a signed coin by simpy reblinding the coin and the signature - the fact that the coin has a special structure prevents this from working.

The one-way coin function can, in principle, be any one way function producing a sufficient number of bits (i.e. around the same number as in p), but the one chosen for Lucre is defined as follows: Let the random seed for the coin be in $[0, 2^n)$ where

$$n = m + ((\log_2(p) - m) \bmod 160) \tag{59}$$

m is the minimim number of bits in x, chosen to be large enough to avoid collisions (128 in Lucre's case). We then define

$$\text{oneway}(x) = x \| SHA1(x\|1) \| SHA1(x\|2) \| \cdots \| SHA1(x\|(\log_2(p) - n)/160) \tag{60}$$

where $\|$ denotes concatenation. In case it isn't obvious, this ensures that

$$\log_2(\text{oneway}(x)) \approx \log_2(p) \tag{61}$$

Note that the resulting coin must actually be in G, so it may take several attempts to find a correct one.

Also note that Lucre encodes the appended numbers as two bytes, LSB first. This is massive overkill and allows for gigantic primes.

9.3 A Possibly Weak One-Way Coin Function

In an earlier version of Lucre, we defined the one-way function like this

$$h_0(x) = x \tag{62}$$
$$h_k(x) = h_{k-1}(x) \| SHA1(h_{k-1}(x)) \tag{63}$$

$$\text{oneway}(x) = h_{(n-m)/160}(x) \tag{64}$$

The problem with this is that

$$\text{oneway}(x|SHA1(x)) = \text{oneway}(x)/2^{160} + O(2^{160}) \tag{65}$$

Whilst we can't see an attack that uses this property, we find it slightly worrying, and so prefer the more conventional construction above.

9.4 A Bad One-way Coin Function

An earlier version of this paper contained an "improvement" to the one-way coin generation - namely the need to test the resulting coin for membership in G was removed by adding an extra step:

$$\text{oneway}(x) = g^{\text{preoneway}(x)} \pmod p \tag{66}$$

where preoneway() is the one-way function defined above. This guarantees the coin is in G. However, the consequences are disastrous[5].

The mint publishes $g^k \bmod p$. A coin's signature is $\text{oneway}(x)^k \bmod p$. But

$$\text{oneway}(x)^k = (g^{\text{preoneway}(x)})^k = (g^k)^{\text{preoneway}(x)} \pmod p \tag{67}$$

That is, the signature can be forged, trivially!

References

[1] Ian Goldberg – mail to *coderpunks*.

[2] David Chaum and Torben Pedersen – "*Wallet databases with observers*" – Advances in Cryptology – Proceedings of Crypto '92, pp. 89-105.

[3] Antoine Joux and Kim Nguyen – "*Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups*" – http://eprint.iacr.org/2001/003.

[4] Ian Goldberg – mail to *coderpunks*.

[5] David Wagner – personal communication.

[6] Anonymous – http://www.mail-archive.com/coderpunks@toad.com/msg02186.html and http://www.mail-archive.com/coderpunks@toad.com/msg02323.html.

10 Change History

A change history was not kept until v1.8. The original version of this paper was written 30 November 1999.

10.1 Changes in 1.8

Released 1 June 2003.

Several errors in the handling of exponents in modulo arithmetic and checks for invertability were corrected.