Digital Cash

is not a check, credit card or a debit card. They leave audit trails.

is anonymous and untraceable.

can be sent through computer networks.

can be used off-line (not connected to a bank).

is transferable.

can be divided (can make change)

can be stolen

can be spent only once (is secure)

might be used to pay for small things (tolls, food).

Protocol 0.

1. Bank gives Alice a note for \$1000 (like a money order or cashier's check) and subtracts \$1000 from Alice's bank account

2. Alice spends the note with a merchant.

3. Merchant deposits the note in his bank account.

4. Merchant's bank clears note with Alice's bank.

Problems: The note is electronic. It can be easily copied, so Alice could spend it twice. So could the merchant. Also Alice could be traced if the bank remembered the serial number of the note.

We solve these problems one-by-one.

Protocol 1.

1. Alice prepares 100 anonymous money orders for \$1000 each.

2. Alice blinds (enciphers) each one and gives all to her bank.

3. Bank asks Alice to open (decipher) 99 envelopes, and verifies that each one is a money order for \$1000. (If not so, Alice goes to jail.)

4. Bank signs the last one, returns it to Alice and deducts $1000 from her bank account.

5. Alice deciphers the money order (now signed by the bank) and spends it with a merchant.

6. Merchant verifies bank's signature (can be done without communication with bank) to make sure it is valid.

7. Merchant takes money order to his bank.

8. Merchant's bank verifies the signature and adds $1000 to merchant's bank account.

This protocol makes anonymous cash, but cash that can still be spent twice.

How does Alice's bank "sign" her enciphered money order?

Let's say the bank uses RSA with keys n, e, d.

Let the money order be M.

Alice chooses a random k in $1 < k < n$.

Alice "blinds" (enciphers) the money order by computing $t = Mk^e \bmod n$.

Bank signs t as

$$t^d \equiv (Mk^e)^d \equiv M^D k^{ed} \equiv M^d k \pmod{n}.$$

Alice "unblinds" (deciphers) the signed money order t^d by computing $k^{-1} \bmod n$ (via extended Euclid) and multiplying:

$$s = t^d k^{-1} \bmod n \equiv (Mk^e)^d k^{-1} \equiv$$

$$\equiv M^d k^{ed-1} \equiv M^d \pmod{n}.$$

To “open” (decipher) the 99 t, Alice tells the bank M and k for each. The bank verifies that each $t = Mk^e \bmod n$.

Protocol 2.

1. Alice prepares 100 anonymous money orders for \$1000 each. On each one she writes a random 20-digit integer.

2. Alice enciphers each one and gives all to her bank.

3. Bank asks Alice to open (decipher) 99 envelopes, and verifies that each one is a money order for \$1000 and that all 99 20-digit integers differ. (If not so, Alice goes to jail.)

4. Bank signs the last money order, returns it to Alice and deducts \$1000 from her bank account.

5. Alice deciphers the money order (now signed by the bank) and spends it with a merchant.

6. Merchant verifies bank's signature to make sure it is valid.

7. Merchant takes money order to his bank.

8. Merchant's bank verifies the signature, and checks in a database to make sure that a money order with the same 20-digit integer has not been previously spent. If this has not happened, then it adds $1000 to the merchant's bank account and records the 20-digit number in the database used by all banks. But if the number is already in the database, then the bank doesn't accept the money order.

Now if Alice tries to spend the money order twice, or if the merchant tries to deposit it twice (in two different banks, say), the second bank will know and not accept it.

The next protocol tries to identify the cheater.

Protocol 3.

1. Alice prepares 100 anonymous money orders for $1000 each. On each one she writes a random 20-digit integer.

2. Alice enciphers each one and gives all to her bank.

3. Bank asks Alice to open (decipher) 99 envelopes, and verifies that each one is a money order for $1000 and that all 99 20-digit integers differ. (If not so, Alice goes to jail.)

4. Bank signs the last money order, returns it to Alice and deducts $1000 from her bank account.

5. Alice deciphers the money order (now signed by the bank) and spends it with a merchant.

6. Merchant verifies bank's signature to make sure it is valid.

7. Merchant asks Alice to write a random identity string on the money order.

8. Alice does so.

9. Merchant takes money order to his bank.

10. Merchant's bank verifies the signature, and checks in a database to make sure that a money order with the same 20-digit integer has not previously deposited. If it hasn't, the bank credits $1000 to the merchant's account and records the 20-digit integer and the random identity string in the database.

11. If the 20-digit integer is already in the database, the bank refuses the money order. Then it compares the identity string on the money order with the one in the database. If they are the same identity string, the bank knows it was the merchant who copied the money order (and he goes to jail). If they differ, the bank knows that the person who bought the money order from a bank copied it.

But the bank doesn't know in the latter case that the person was Alice. The next protocol repairs this flaw.

Secret splitting (used in the next protocol):

Let the secret be s.

Choose a random r of the same length as s.

One party gets r; the other gets $r \oplus s$.

Together the two parties can compute

$$r \oplus (r \oplus s) = s.$$

Bit commitment (used in the next protocol)

Alice wants to commit to a "bit" (it can be a string) now so that she can't change it later, but only Alice knows it for now.

Alice commits to b by generating two random strings R_1, R_2.

She creates a message (R_1, R_2, b) and computes a hash (SHA. say) of it.

She reveals (or tells Bob) the hash value and R_1.

When the time comes to reveal b, Alice shows the message (R_1, R_2, b).

Bob checks that R_1 is the same as it was earlier and verifies the hash value.

Why is R_2 needed? Because if b were a short string (one bit, literally, say), then Bob could guess it from the hash value.

Protocol 4.

1. Alice prepares 100 anonymous money orders for \$1000 each. On each one she writes a random 20-digit integer and 100 pairs of identity bit strings: $(I_{1L}, I_{1R}), \ldots, (I_{100L}, I_{100R})$. Each part is a bit-committed packet that Alice can be asked to open, and whose proper opening can easily be checked. Any pair, (I_{59L}, I_{59R}), say, reveals Alice's identity by secret splitting, that is,

$$I_{59L} \oplus I_{59R} = \text{Alice}'\text{s identity.}$$

2. Alice enciphers each money order and gives all 100 to her bank.

3. Bank asks Alice to open (decipher) 99 envelopes, and verifies the contents. If it finds an error, Alice goes to jail.

4. Bank signs the last money order, returns it to Alice and deducts $1000 from her bank account.

5. Alice deciphers the money order (now signed by the bank) and spends it with a merchant.

6. Merchant verifies bank's signature to make sure it is valid.

7. Merchant asks Alice to randomly reveal either the left or right half of each of the 100 identity strings. (Merchant chooses the random choices of left or right half.)

8. Alice complies.

9. Merchant takes money order to his bank.

10. Bank verifies the signature and checks the database for the 20-digit number. If it is not found therein, the bank credits the merchant with $1000 and records the money order in the database.

11. If the 20-digit integer is already in the database, the bank does not accept the money order. It compares the 100 identity strings on the money order with those in the database. If they agree, the bank knows that the merchant copied the money order. If they differ, a second merchant deposited the money order earlier and it was Alice who copied. Some of the 100 identity strings will have both halves revealed, so Alice can be identified. Alice goes to jail.

Remarks on Protocol 4:

It is not transferable or divisible.

Can Alice cheat? She can copy her $1000 money order. The first time she spends it is okay. But she gets caught the second time she spends it.

Can she create a money order with a bad id string? One chance in 100.

Alice can't change the 20-digit number or the identity strings, because then the bank's signature would no longer be valid.

Can the merchant cheat? No.

Can the merchant and Alice collude to spend the e-cash twice? No, because they can't change the 20-digit number signed by the bank, so the bank will not have to pay the $1000 more than once.

Can Eve copy Alice's money order and spend it first? Yes. It is like cash.

Even worse, if Alice didn't know that Eve copied it and spent it, then Alice would be caught when she spent it the first time.

Eve could eavesdrop on communication between Alice and the merchant and deposit the money (acting as a merchant) before the merchant deposits it. When the merchant tries to deposit it, he will be found as a cheater.

Both Alice and the merchant must protect their e-cash as if it were cash. It must be enciphered when it is send across the Net.

The Perfect Crime

1. Alice kidnaps a baby.

2. Alice prepares 10,000 anonymous money orders for $1000 each.

3. Alice blinds them, sends them to authorities, and demands that

a. a bank signs all 10,000 money orders, and

b. the results be published in a newspaper.

4. The authorities comply.

5. Alice buys the newspaper, unblinds all the money orders and spends them.

6. Alice frees the baby.

Note that there is no physical contact.