Question 1: Sources of Randomness

Which of the following are good sources of randomness for key generation and which for IV generation (CBC-mode)? What is the reason?

- computer time in seconds since UNIX epoch
- hardware random number generator
- coin flips from a fair coin
- coin flips from a biased coin that is 90% of the times 'heads'.
- a true random page number from a phone book
- a random last digit of a random phone number from a phone book.
- a random first digit of a random phone number from a phone book.
- parent process ID $\oplus$ my process ID $\oplus$ seconds since UNIX epoch
- bytes from `/dev/random`

Question 2: Prediction and Rollback Resistant

For the following PRNG, do they have prediction resistance and do they have rollback resistance? Assume that standard cryptographic assumptions hold and that any seed values come from true randomness.

Scheme 1

```
// helper functions

// returns sha256 digest of message
string sha256(message);

// PRNG state
struct state_t {
        string cur_s;
}

// PRNG seeding
state_t init_prng(char seed[4]) {
        state_t ret;
        ret.cur_s = sha256(seed);
        return ret;
}
```

```
// PRNG randomness function and state advancement
string next_prng(struct state_t& state) {
 state.cur_s = sha256(state.cur_s);
 return sha256(state.cur_s + "extra");
}
```

Scheme 2

```
// helper functions

// returns sha256 digest of message
string sha256(message);

// PRNG state
struct state_t {
        string cur_s;
        uint128_t cur_i;
}

// PRNG seeding
state_t init_prng(char seed[16]) {
        state_t ret;
        ret.cur_s = sha256(seed);
        ret.cur_i = 0;
        return ret;
}

// PRNG randomness function and state advancement
string next_prng(struct state_t& state) {
        ++state.cur_i;
        for (size_t i = 0; i < state.cur_i; ++i) {
                ret = sha256(state.cur_s);
        }
        return ret;
}
```

Question 3: Adobe Password Breach

The 2013 Adobe password breach effect dozens of millions of active users. Adobe stored a database that contained a user's login email, their password, and a free-form password hint they were allowed to set with their password. This database was leaked.

Adobe took the precaution of "encrypting" the user's password, which they did with a block cipher in ECB mode. The key did not leak with the database.

A snippet of the data is below. Observe that the encrypted password is encoded in Base64.

```
adalberto@wtinformatica.com.br:x1NEI6ID10/ioxG6CatHBw==:moradia
williambaltzegar@att.net:f7MjFoPc+2HioxG6CatHBw==:golf score & age
studio@stevenlyons.com:2GOFpSIZT6w=:bd
siebenal@execpc.com:qFTW7ejjck4=:len
rick@two-five.com:4FmFjqE/99PioxG6CatHBw==:dogs name
ron@pldev.com:N+Iv46afuHk=:bike
gary@orb.com.au:d2t4QxOV+Sg5IQsp4TdDow==:usual
```

1. We discourage ECB mode for what reason? How does this problem manifest here? How does the scale of the database exacerbate it?

2. Users were free to provide a password hint, that is, some phrase that provides a hint to their password. Even if a particular hint doesn't clearly reveal the password, what can still be done to recover plaintext passwords?

After you've worked through this question, check out `https://zed0.co.uk/crossword/` for a demo of why ECB mode is bad.