Lecture 6: Modes of Operation and Random Numbers

TTM4135

Relates to Stallings Chapters 7 and 8

Spring Semester, 2025

Motivation

- Block ciphers encrypt single blocks of data but in applications many blocks of data are encrypted sequentially
- The simple approach to break up the plaintext into blocks and encrypt each separately is generally insecure
- There are many different *modes of operation* which are standardised with different security and efficiency
- Random numbers are needed in many uses of cryptography
- Block ciphers can be used to generate random numbers

Outline

Purpose of different modes

- Modes can be designed to provide *confidentiality* for data, or to provide *authentication* (and integrity) for data or to provide both
- In this lecture we focus on **confidentiality** modes which normally must include **randomisation**
- Some modes can be used to generate pseudorandom numbers
- Different modes have different efficiency properties or communications properties

Reminder from Lecture 3: Vigenère cryptanalysis

The first characters of a ciphertext are:

AUVHSGE**PELPEK**QTEDKSFNYJYATCTCCKFTSUTEFVBVVHPNMFUHBFNPV
YFVRFVUSPEEVHFNAOFLBFYJPFPTMFFMFVHBVHFJAENEGVTIGHPWSFU
HPTTMAAGVESGIHJT**PELPEK**JPTIGMPTNJPGJUAUFOXPBFUIEGTIGFJTEIQ
WFXESYIUJTIGIOVEOVIPPOGCWBKTJPGIKMIQWFXESNOOIHFOIHJTCGIXC
SBNRFCDZFEFRLZKNUGRFUTFFIOJITKNRWISAFPTTIQUHJIUYATUUSTOVP
DFFBZPOOGOGVHFIRJOAOFSUTAOIEGGAUWRFUWIKCIYESGATUODKAUG
DXKTIVHFVWPERJOETYHJEHJJAWGAMTEBFYSGCPTDFFSUKLMVHFPAUW
RFQFUJEDCSFCNEVHFGXBNTFFSUCTJQNPHHJUCMKEOVGBXEJVADJASC
CUGRPHIUUOXPIOFEFFAQCRUHRPOTIGNBVUSGOGVHFKNWGSUKGBVIP
P**WIKCIO**YGTIFPDICDPPHBPDUJESGWBUSPOEUJIOIIOJITOATVESNYHTATR
OGCSJVUBVIPPAOFHJUKFGNJPCJUIWGRFCSPPIOI**WIKCIO**AEGIUCPMGAT
WRFVONGTPUTVFYIKSTASUGMPHWPTKBPDUQFPNLPYTIGQVKCLUUCVL
FOEUJOEUBZYHJEHIGDJUEOVAOILFFTIGMPUTJPEYVRJEACNENASUGRJ

The importance of randomised encryption

- It is a problem if the same plaintext block is encrypted to the same ciphertext block every time. This would allow patterns to be found in a long ciphertext.
- We *randomised* encryption schemes to prevent this.
- Typically this is achieved using an *initialisation vector*, IV, which propagates through the entire ciphertext. The IV may need to be:
  - unique;
  - and/ or random.
- Another way to vary the encryption is to include a variable state which is updated with each block.

Efficiency

There are a number of important features of different modes which do not impact security but are really important for practical usage.

- Some modes allow *parallel processing*:
  - sometimes multiple plaintext blocks can be encrypted in parallel;
  - sometimes multiple ciphertext blocks can be decrypted in parallel.
- Some modes result in *error propagation*: a bit error which occurs in the ciphertext results in multiple bit errors in the plaintext after decryption.

Padding

- Some modes, including ECB and CBC, require the plaintext to consist of one or more complete blocks.
- NIST Special Publication 800-38A suggests a padding method as follows:
  1. append a single ‘1’ bit to the data string
  2. pad the resulting string by as few ‘0’ bits, possibly none, as are necessary to complete the final block.
- The padding bits can be removed unambiguously, if the receiver knows that this padding method is used:
  1. remove all trailing ‘0’ bits after the last ‘1’ bit
  2. remove a single ‘1’ bit.
- An alternative to padding is *ciphertext stealing* (see exercises).

Notation overview

- The message is n blocks in length
- P represents the plaintext message
- C represents the ciphertext message
- P_t represents plaintext block t where $1 \leq t \leq n$
- C_t represents ciphertext block t where $1 \leq t \leq n$
- K represent the key
- IV represents the initialisation vector

All modes can be applied to any block cipher. A case of special interest is AES when blocks are 128 bits in length.

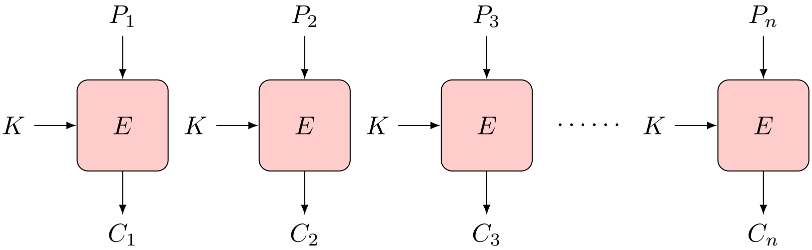
NIST Standards

- Four modes ECB, CBC, CFB and OFB were originally standardised for use with DES in 1980. CTR mode was added in 2001, initially for use with AES.
- SP 800-38A (2001) Confidentiality Modes: ECB, CBC, CFB and OFB. An addendum defines Ciphertext Stealing
- SP 800-38B (2016) CMAC Mode for Authentication
- SP 800-38C (2004, updated 2007) CCM Mode
- SP 800-38D (2007) Galois/Counter Mode (GCM)
- SP 800-38E (2010) XTS-AES Mode for Storage Devices
- SP 800-38F (2012) Key Wrapping
- SP 800-38G (2016) Format-Preserving Encryption

Electronic Code Book (ECB) mode

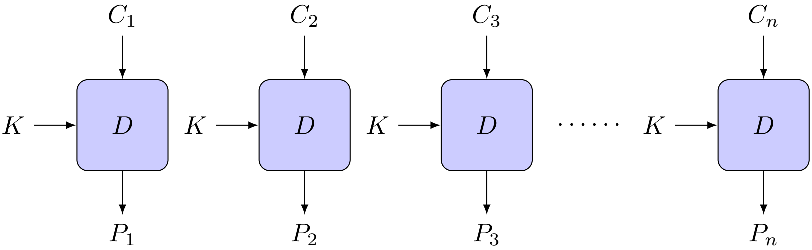
- ECB is the basic mode of a block cipher
- *Encryption*:
  - $C_t = E(P_t, K)$
  - Plaintext block P_t is encrypted with the key K to produce ciphertext block C_t
- *Decryption*:
  - $P_t = D(C_t, K)$
  - Ciphertext block C_t is decrypted with the key K to produce plaintext block P_t

ECB mode encryption



- Blocks $C_1, C_2, \ldots C_n$ are sent

ECB mode decryption



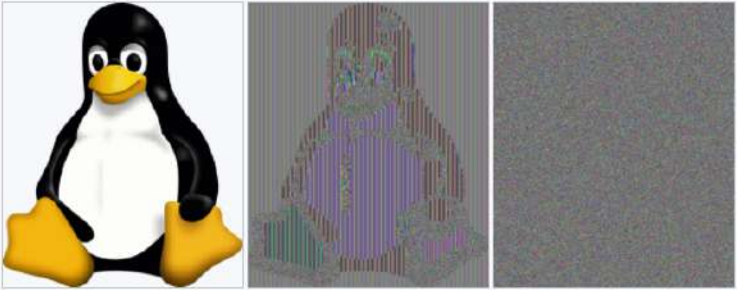
- Blocks $C_1, C_2, \ldots C_n$ are received

ECB mode properties

Randomised	✘
Padding	Required
Error propagation	Errors propagate within blocks
IV	None
Parallel encryption?	✓
Parallel decryption?	✓

- Because it is deterministic, ECB mode is not normally used for bulk encryption.
- If a block is repeated in the plaintext, it will be repeated in the ciphertext!
- Encrypting with ECB mode may reveal *patterns* in the plaintext.

ECB mode – weakness



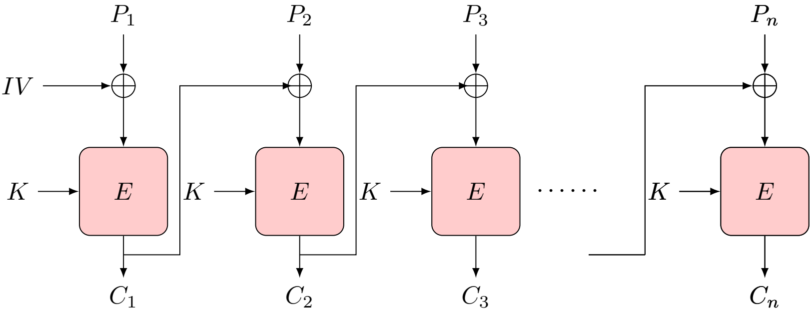
Original image | Using ECB allows patterns to be easily discerned | Modes other than ECB result in pseudo-randomness

Source: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Cipher Block Chaining (CBC) mode

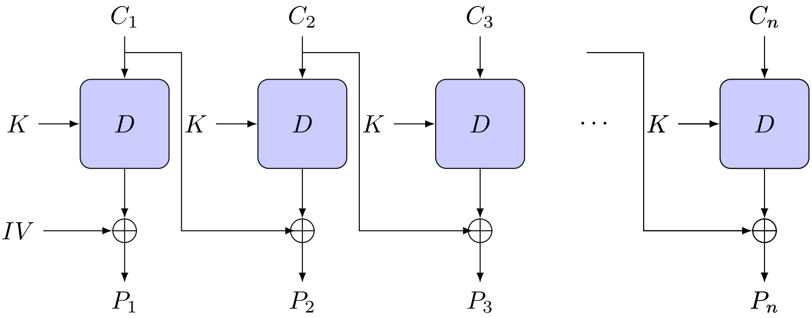
- CBC "chains" the blocks together
- A random initialisation vector IV is chosen and sent together with the ciphertext blocks
- *Encryption*:
  - $C_t = E(P_t \oplus C_{t-1}, K)$, where $C_0 = IV$
  - P_t is XOR'd with the previous ciphertext block C_{t-1}, and encrypted with key K to produce ciphertext block C_t
  - IV is used for the value C_0 and sent with $C_1, \ldots C_n$
- *Decryption*:
  - $P_t = D(C_t, K) \oplus C_{t-1}$, where $C_0 = IV$
  - C_t is decrypted with the key K, and XOR'd with the previous ciphertext block C_{t-1} to produce plaintext block P_t
  - As in encryption, C_0 is used as the IV

CBC mode encryption



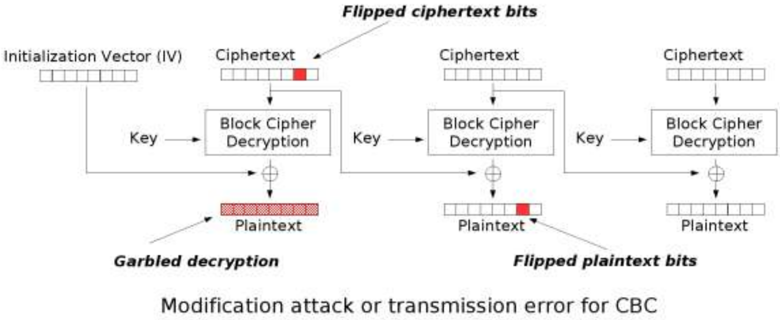
- IV and blocks $C_1, C_2, \ldots C_n$ are sent

CBC mode decryption



- IV and blocks $C_1, C_2, \ldots C_n$ are received

CBC mode error propagation



Public domain figure from:

http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

CBC mode properties

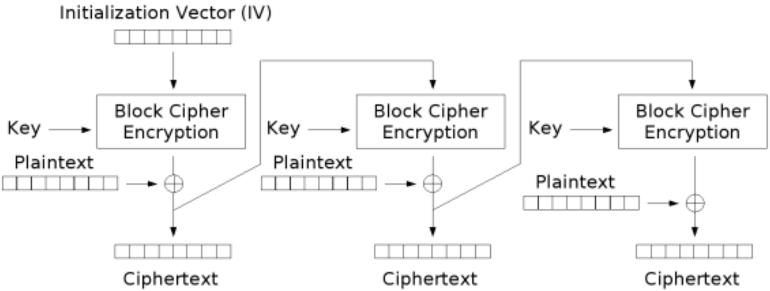
Randomised	✓
Padding	Required
Error propagation	Errors propagate within blocks and into specific bits of next block
IV	Must be random
Parallel encryption?	✗
Parallel decryption?	✓

- Commonly used for bulk encryption
- Common choice for channel protection in all versions of TLS up to TLS 1.2

CFB mode

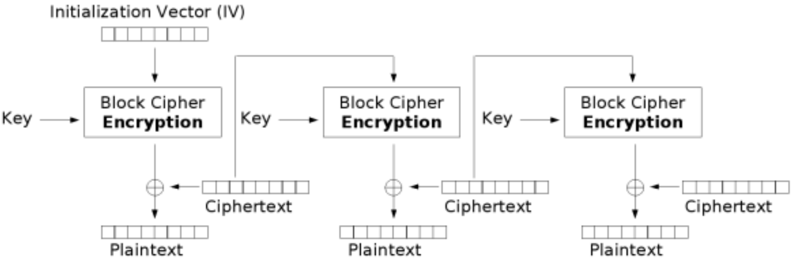
- CFB “feeds” the ciphertext block back into the enciphering/deciphering process, thus “chaining” the blocks together.
- *Encryption*:
  - $C_t = E(C_{t-1}, K) \oplus P_t$, where $C_0 = IV$
- *Decryption*:
  - $P_t = E(C_{t-1}, K) \oplus C_t$
- *Propagation of channel errors*
  - a one-bit change in C_t produces a one-bit change in P_t, and complete corruption of P_{t+1}

CFB mode encryption



Cipher Feedback (CFB) mode encryption

CFB mode decryption



Cipher Feedback (CFB) mode decryption

Self synchronisation in CFB mode

- CFB is a *self-synchronising stream cipher*:
  - Keystream depends on previous ciphertexts, which allows CFB mode to self-synchronise after processing a correct ciphertext block.
  - Assume block C_t is lost in transmission, producing a loss in synchronicity between sender and receiver.
  - Receiver decrypts next received block C_{t+1} as $E(C_{t-1}, K) \oplus C_{t+1}$, which is incorrect (i.e. different from P_t).
  - For the next received block C_{t+2} the receiver computes $E(C_{t+1}, K) \oplus C_{t+2}$, which is the correct plaintext block P_{t+2}. The cipher is back in sync (after losing P_t and P_{t+1}).
- CFB mode can also be defined with a sub-block feedback. In this case re-synchronisation can occur after loss of a sub-block.

CFB mode properties

Randomised	✓
Padding	Not required
Error propagation	Errors occur in specific bits of current block and propagate into next block
IV	Must be random
Parallel encryption?	✗
Parallel decryption?	✓

CFB mode is commonly used when self-synchronisation is useful.

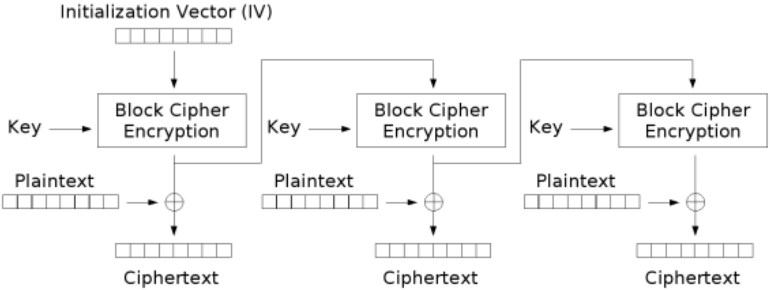
OFB mode

- OFB “feeds” the output block back into enciphering/deciphering process.
- OFB is, in effect, a *synchronous stream cipher*. The keystream is:

$$O_t = E(O_{t-1}, K),$$

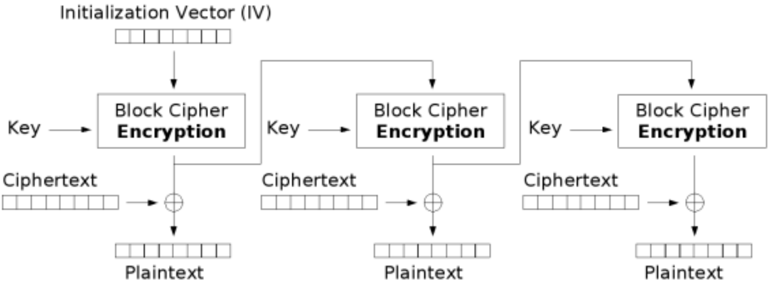
  where $O_0 = IV$ is chosen at random.
- *Encryption*:
  - $C_t = O_t \oplus P_t$
- *Decryption*:
  - $P_t = O_t \oplus C_t$
- *Propagation of channel errors:*
  - a one-bit change in the ciphertext produces a one-bit change in the plaintext at the same location

OFB mode encryption



Output Feedback (OFB) mode encryption

OFB mode decryption



Output Feedback (OFB) mode decryption

OFB mode properties

Randomised	✓
Padding	Not required
Error propagation	Errors occur in specific bits of current block
IV	Must be unique
Parallel encryption?	✗ (but keystream can be computed in advance)
Parallel decryption?	✗

OFB mode is a synchronous stream cipher mode.

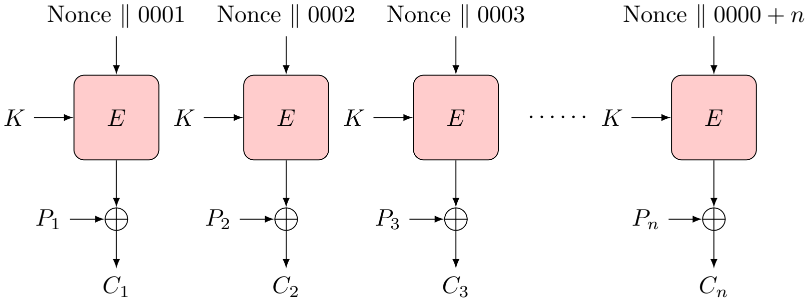
Counter (CTR) mode

- CTR is a *synchronous stream cipher*. The keystream is generated by encrypting successive values of a "counter", initialised using a nonce (randomly chosen value) N:

$$O_t = E(T_t, K),$$

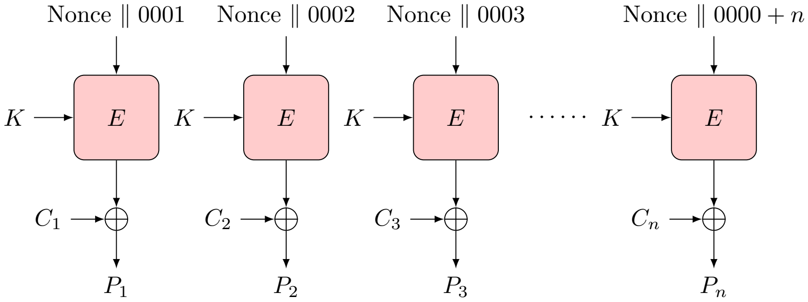
  where $T_t = N \| t$ is the concatenation of the nonce and block number t.
- *Encryption*:
  - $C_t = O_t \oplus P_t$
- *Decryption*:
  - $P_t = O_t \oplus C_t$
- *Propagation of channel errors:*
  - a one-bit change in the ciphertext produces a one-bit change in the plaintext at the same location

CTR mode encryption



- Nonce and blocks $C_1, C_2, \ldots C_n$ are sent

CTR mode decryption



- Nonce and blocks $C_1, C_2, \ldots C_n$ are received

CTR mode properties

Randomised	✓
Padding	Not required
Error propagation	Errors occur in specific bits of current block
IV	Nonce must be unique
Parallel encryption?	✓
Parallel decryption?	✓

- A *synchronous stream cipher mode*
- Good for access to specific plaintext blocks without decrypting the whole stream
- Basis for authenticated encryption in TLS 1.2 and TLS 1.3

Principles of (pseudo)random number generation

- Random numbers play a crucial role in cryptography – without strong randomness, we do not have cryptography
  - Think of e.g. encryption keys
- Usually we define some statistical notion. In particular, the two following criteria are important.
  - **Uniform distribution:** I.e. the frequency of ones and zeroes should be approximately equal (for binary outputs).
  - **Idependence:** No output should be predictable given previous/ future outputs.

Randomness in cryptography

- Truly random numbers are used in some applications, but they have drawbacks, such as inefficiency.
- Thus, it is more common to generate sequences of numbers that *appear to be random* but are not random
- In particular, we must ensure that *an adversary cannot predict* future elements
- Cryptographic applications typicaly make use of algorithmic techinques for random number generation.
- These algorithms are *deterministic*, and therefore produce sequences that are not statistically random.
- We refer to these numbers as *pseudorandom* numbers.

Randomness

- Any specific string of bits (number) is exactly as random as any other string – which is why we look at *distributions* (in a statistics sense)
- We think instead in terms of *generators* of random strings
  - A *true random number generator* (TRNG) is a physical process which outputs each valid string independently with equal probability
  - A *pseudo random number generator* (PRNG) is a deterministic algorithm which approximates a TRNG
- We may use a TRNG to provide a *seed* for a PRNG

TRNGs

- NIST Special Publication SP 800-90B (2018) provides a framework for design and validation of TRNG algorithms called *entropy sources*.
- The entropy source includes a physical noise source, a digitization process and post-processing stages. The output is any requested number of bits.
- The standard specifies many statistical tests for validating the suitability of entropy sources
- An important additional requirement is a periodic *health test* to ensure continuing reliable operation
- Intel introduced TRNGs into Ivy Bridge processors in 2012

PRNGs

- NIST Special Publication SP 800-90A (June 2015) recommends specific PRNG algorithms named *Deterministic Random Bit Generators* (DRBG) based on:
  - hash functions (we look at these in a later lecture);
  - a specific MAC known as HMAC (also in a later lecture);
  - block ciphers in counter mode.
- Each generator takes a seed as input and outputs a bit string before updating its state
- The seed should be updated after some number of calls
- The seed can be obtained from a TRNG

TRNGs, PRNGs and PRFs

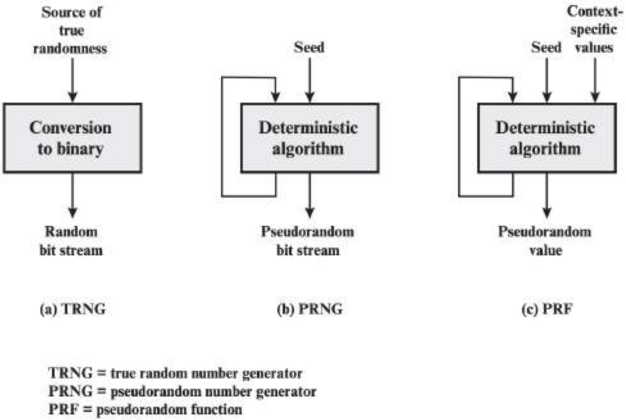


Figure 8.1 Random and Pseudorandom Number Generators

Cryptography and Network Security, Stallings

TRNGs input into PRNGs

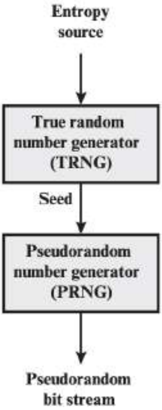


Figure 8.2 Generation of Seed Input to PRNG

Cryptography and Network Security, Stallings

Functions of DRBGs

The SP 800-90A standard defines a general model for DRBGs with some functions

Instantiate This sets the initial state of the DRBG using a seed

Generate This provides an output bit string for each request

Reseed Inputs a new random seed and updates the state

Test Checks correct operation of the other functions

Uninstantiate Deletes (“zeroises”) the state of the DRBG

Security of DRBGs

The standard defines the security of DRBGs in terms of the ability of an attacker to *distinguish reliably* between its output and a truly random string. Two properties are defined.

Backtracking resistance
: An attacker who obtains the current state of the DRBG should not be able to distinguish between the output of earlier calls to the DRBG generate function and random strings.

Forward prediction resistance
: An attacker who obtains the current state of the DRBG should not be able to distinguish between the output of later calls to the DRBG generate function and random strings.

CTR_DRBG

- Uses a block cipher in CTR mode. AES with 128-bit keys is one recommended option.
- The DBRG is initialised with a seed whose length is equal to the key length plus the block length, so $128 + 128 = 256$ bits for AES with 128-bit keys.
- From a high entropy seed a key K and state (counter) value V are derived. There is no separate nonce as in normal CTR mode.
- Counter mode encryption is then run iteratively (with no plaintext added) and the output blocks form the output.

Update function in CTR_DRBG

- The Update function is used in the Initialise, Generate and Reseed functions to **generate new key and state**
- Inputs are current key K and state (counter) V and optional data input D
- Output is a new key K' and state (counter) V'
- When block and key size are the same, computation is:
  - Generate new block $O_1 = E(V, K)$
  - Increment V
  - Generate new block $O_2 = E(V, K)$
  - $K' \parallel V' = (O_1 \parallel O_2) \oplus D$
- Updating provides backtracking resistance

Instantiate, generate and reseed in CTR_DRBG

Instantiate calls the Update function with D equal to high entropy seed, and K and V all zero strings

Generate computes up to 2^{19} bits by running CTR mode output from the current state. Then the Update function is called with D empty

Reseed calls the Update function with D equal to high entropy input, and K and V in the current state.

- The standard restricts the output to 2^{48} requests to the Generate function before Reseed must be called
- Each Reseed call provides forward prediction resistance and backtracking resistance

Dual_EC_DRBG

- Older standard (December 2012) includes Dual_EC_DRBG
- Based on elliptic curve discrete logarithm problem (we look at this in a later lecture)
- Much slower than other DRBGs in the standard
- Based on hard problem but no security proof exists. In fact NTNU's Kristian Gjøsteen and others showed in 2006 that the output has an observable bias
- In December 2013 the press reported a secret ten million dollar deal between the NSA and RSA Security company to use Dual_EC_DRBG as the default PRNG in its software suite

`http://blog.cryptographyengineering.com/2013/09/the-many-flaws-of-dualecdrbg.html`

Cloudfare

- Cloudfare is a company located in San Francisco, USA
- They provide:
  - Content delivery network services
  - Cloud cybersecurity
  - DDoS mitigation
- Cloudflare is used by more than 20 percent of the Internet for its web security services as of 2022.

https://en.wikipedia.org/wiki/Cloudflare
https://www.cloudflare.com/en-gb/

How to generate randomness?

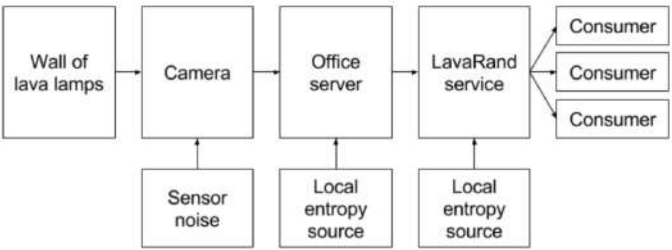
- Cryptographically-secure pseudorandom number generators (CSPRNGs) are algorithms which, provided an input which is itself unpredictable, produce a much larger stream of output which is also unpredictable
- But only half of the equation – they need an unpredictable input
- High-accuracy measurements are unpredictable! e.g. temperature

LavaRand

https://blog.cloudflare.com/randomness-101-lavarand-in-production

LavaRand

The pipeline – LavaRand is only a backup



https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details