Public

Defense in Depth - The Path to SGX at Akamai

Samuel Erb -- serb@akamai.com
@erbbysam

DEFCON 26 PACKET HACKING VILLAGE

whoami

Samuel Erb -- @erbbysam

Software Engineer at Akamai

DC23, DC24 black badge (Badge Challenge, Co9)

(disclaimer)

Nothing presented here gives you permission to poke and prod at Akamai.

Everything presented here is the work of **many** individuals within Akamai.

Defense in Depth

Dictionary

defense in depth

defense in depth

phrase of defense

1. the practice of arranging defensive lines or fortifications so that they can defend each other, especially in case of an enemy incursion.

Translations, word origin, and more definitions

Feedback

Defense in depth (computing) - Wikipedia

https://en.wikipedia.org/wiki/Defense_in_depth_(computing)

Defense in depth (also known as Castle Approach) is an information assurance (IA) concept in which multiple layers of security controls (**defense**) are placed throughout an information technology (IT) system.

Background · Controls

Defense in Depth (My Definition)

1. Avoid the next TLS heartbleed at Akamai
2. Put systems in place to accomplish #1

Heartbleed?

- OpenSSL users were vulnerable to Heartbleed

https://blogs.akamai.com/2014/04/heartbleed-a-history.html

```
▼ Secure Sockets Layer
  ▼ TLSv1.2 Record Layer: Heartbeat Request
      Content Type: Heartbeat (24)
      Version: TLS 1.2 (0x0303)
      Length: 16368
    ▼ Heartbeat Message
        Type: Request (1)
      ▶ Payload Length: 16365 (invalid, using 16349 to decode payload)
        Payload (16349 bytes)
        Padding and HMAC (16 bytes)
```

```
RFC 6520              TLS/DTLS Heartbeat Extension          February 2012

   When a HeartbeatRequest message is received and sending a
   HeartbeatResponse is not prohibited as described elsewhere in this
   document, the receiver MUST send a corresponding HeartbeatResponse
   message carrying an exact copy of the payload of the received
   HeartbeatRequest.
```


Heartbleed image via https://commons.wikimedia.org/wiki/File:Heartbleed.svg

TLS?

```
RFC 5246                        TLS                        August 2008

      Client                                               Server

      ClientHello                  -------->
                                                      ServerHello
                                                     Certificate*
                                               ServerKeyExchange*
                                              CertificateRequest*
                                   <--------      ServerHelloDone
      Certificate*
      ClientKeyExchange
      CertificateVerify*
      [ChangeCipherSpec]
      Finished                     -------->
                                               [ChangeCipherSpec]
                                   <--------             Finished
      Application Data             <------->     Application Data

             Figure 1.  Message flow for a full handshake

   * Indicates optional or situation-dependent messages that are not
   always sent.
```

TLSv1.2 base text via https://tools.ietf.org/html/rfc5246

TLS Security Journey

1) **Make secrets harder to find in memory**

OpenSSL Secure Heap

- One of Akamai's contributions to OpenSSL
- Best effort to protect long lived secrets from overruns, underruns, swapping to disk, appearing in coredumps etc
- Utilizes standard Linux memory protection features

more information - https://www.openssl.org/docs/man1.1.0/crypto/OPENSSL_secure_malloc.html

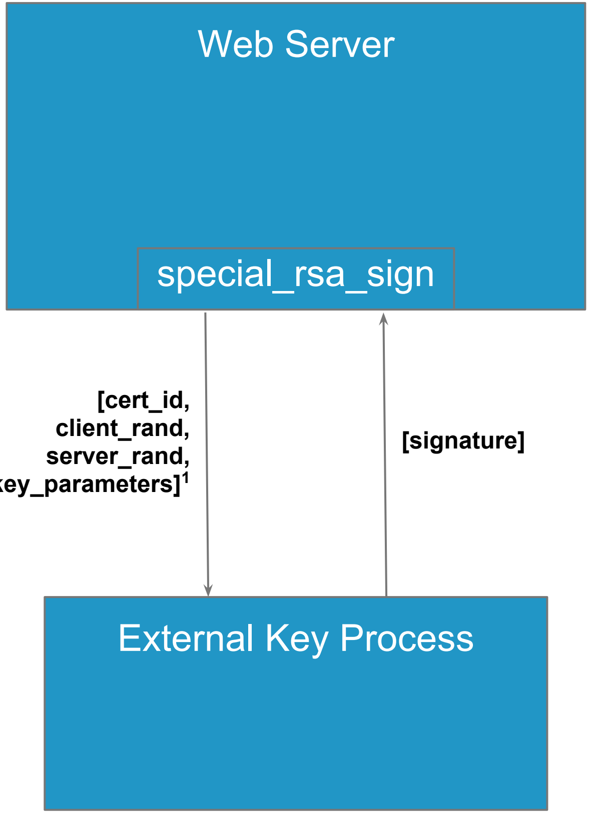
TLS Security Journey

1) Make secrets harder to find in memory
2) **Move the secrets out of the process**

OpenSSL Engine Interface & TLS

```
static int special_rsa_sign(int flen, const unsigned char *from,
                            unsigned char *to, RSA *rsa, int padding)
{

    int ret;

    // handoff to external RSA processor

    return ret;
}

// this is code for openssl > 1.1, previous versions look slightly different
// you may need this -- #if OPENSSL_VERSION_NUMBER >= 0x10100000L
RSA_METHOD *rsa_method = RSA_meth_new("openssl RSA_METHOD",
                                      RSA_METHOD_FLAG_NO_CHECK);
RSA_meth_set_priv_enc(rsa_method, special_rsa_sign);
ENGINE *e = ENGINE_new();
ENGINE_set_id(e, "rsa-engine");
ENGINE_set_name(e, "rsa engine");
ENGINE_set_RSA(e, rsa_method);
ENGINE_add(e);
ENGINE_set_default_RSA(e);
```

OpenSSL Engine Interface & TLS



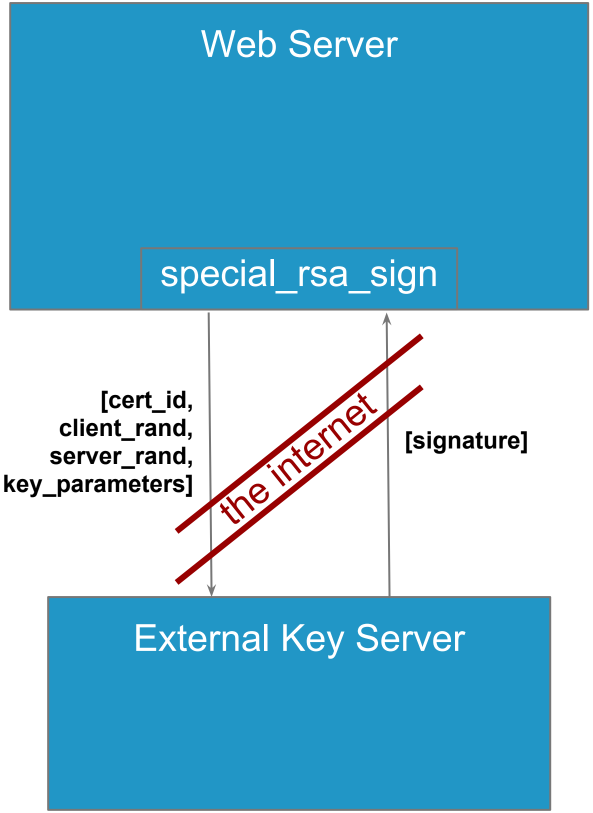

```
static int special_rsa_sign(int flen, const unsigned char *from,
                            unsigned char *to, RSA *rsa, int padding)
{

    int ret;

    // handoff to external RSA processor

    return ret;
}

// this is code for openssl > 1.1, previous versions look slightly different
// you may need this -- #if OPENSSL_VERSION_NUMBER >= 0x10100000L
RSA_METHOD *rsa_method = RSA_meth_new("openssl RSA_METHOD",
                                      RSA_METHOD_FLAG_NO_CHECK);
RSA_meth_set_priv_enc(rsa_method, special_rsa_sign);
ENGINE *e = ENGINE_new();
ENGINE_set_id(e, "rsa-engine");
ENGINE_set_name(e, "rsa engine");
ENGINE_set_RSA(e, rsa_method);
ENGINE_add(e);
ENGINE_set_default_RSA(e);
```

1 - See TLS 1.2 rfc5246 section 7.4.3, “signed_params”

TLS Security Journey

1) Make secrets harder to find in memory
2) Move the secrets out of the process
3) **Move the secrets out of the machine**

Secret Handling Machines



- Protocol is identical to placing secrets in another process
- This creates a many-to-many connection handling & internet routing problem
  - No slow handshakes!
  - There is a performance/security trade off here, at least on the initial TLS handshake

Visualizing a Planetary Scale Network

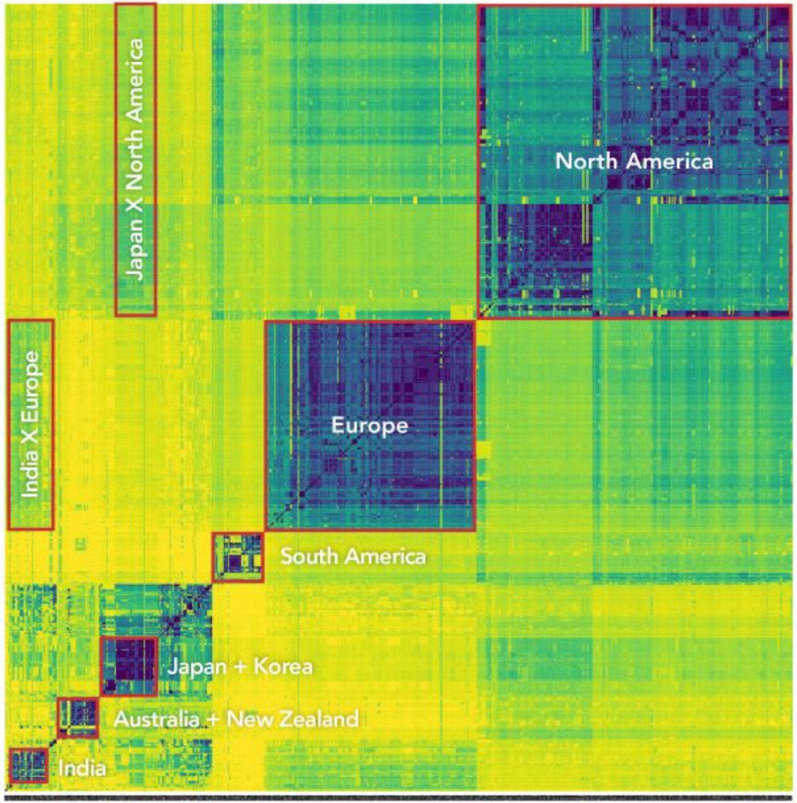
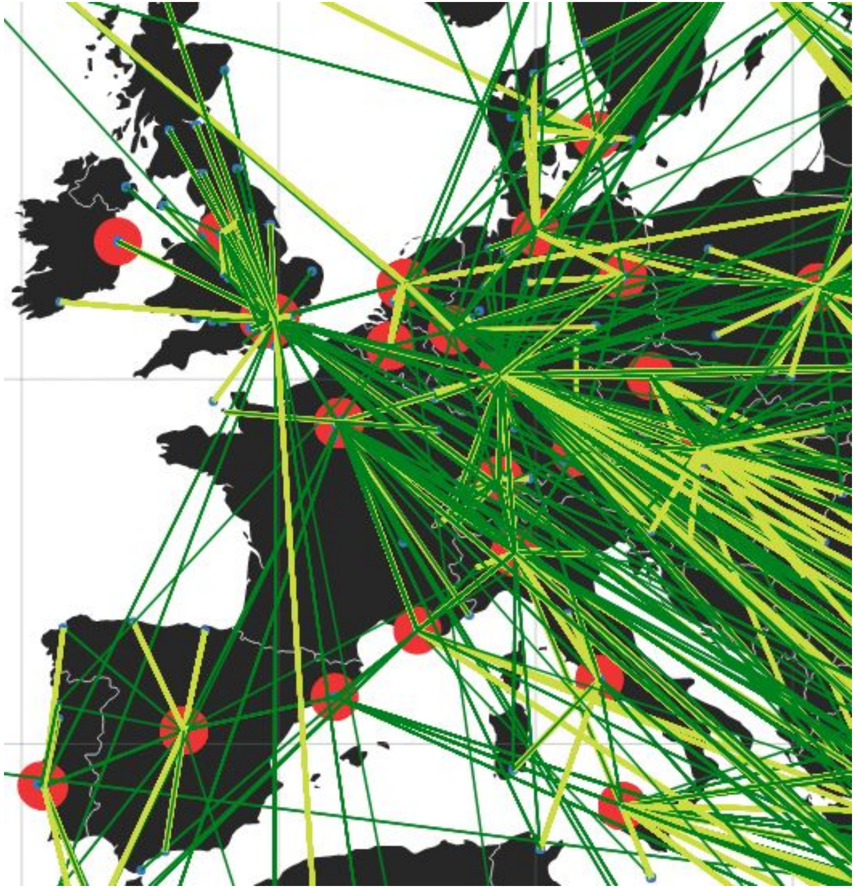


Figure 4-1: Dark colors like purples and blues show short latencies between regions, while light colors (greens and yellows) represent long latencies

via https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q4-2017-state-of-the-internet-security-report.pdf

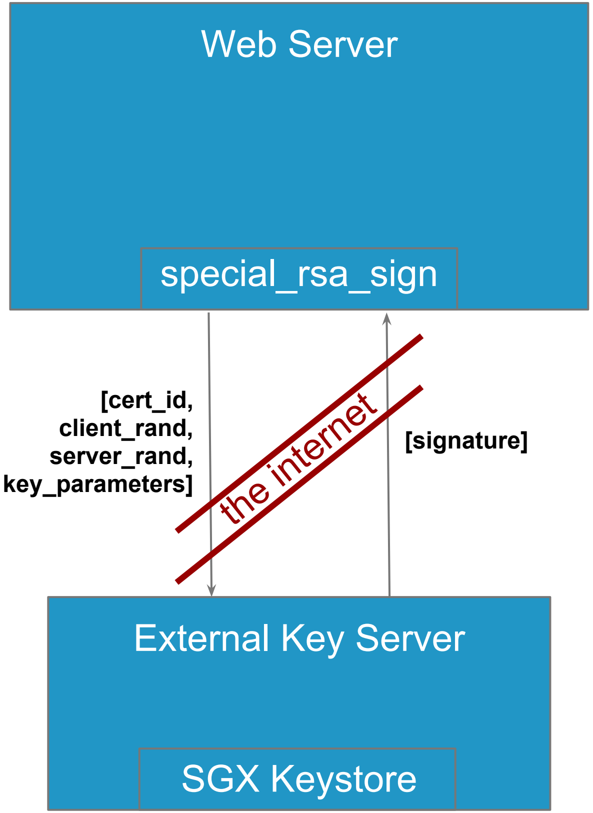
Planetary Scale Network in Practice



TLS Security Journey

1) Make secrets harder to find in memory
2) Move the secrets out of the process
3) Move the secrets out of the machine
4) **Move the secrets into an HSM-like device (Intel SGX)**

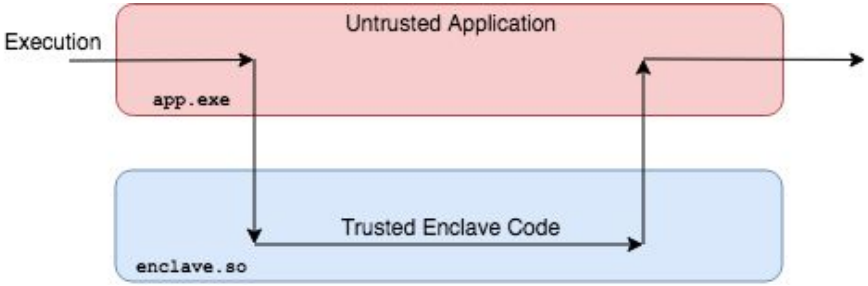
SGX Keystore (HSM-like device)



- Being a bit paranoid -- we've now isolated our secrets, can we remove them from application memory altogether?
- We could also place this SGX keystore inside the web server

What is SGX?

- Software Guard Extensions
- A series of new CPU instructions for handling enclaves
- Enclaves are signed executables that are tightly controlled and cannot be externally monitored
- Entirely on-die - Anything leaving the chip is encrypted
- Extremely limited communication is possible between an enclave and the rest of the system



What is SGX? (continued)

- Reverse sandbox[1]
  - Enclaves can read/write to external memory
  - Applications (at any privilege) cannot read/write to enclave memory
- ~Same performance as CPU itself

1 - term originally coined here: https://www.blackhat.com/docs/us-16/materials/us-16-Aumasson-SGX-Secure-Enclaves-In-Practice-Security-And-Crypto-Review.pdf
image via https://commons.wikimedia.org/wiki/File:Sandpit.jpg

SGX Enclaves (high level)

- Enclaves must be signed. The key used must be trusted by the system.
  - Chain of trust: BIOS trusted “launch enclave” is signed -> your enclave is signed by a key trusted by the “launch enclave” -> “launch enclave” hands out “launch token” allowing your production enclave to start
- Enclaves must be statically compiled
- Requires BIOS & kernel support
- Enclaves cannot interact with anything off-die
- Remote & local attestation is possible

image via https://en.wikipedia.org/wiki/Space_Shuttle_external_tank#/media/File:Space_Shuttle_Columbia_launching.jpg

SGX Enclaves (lower level)

- New instructions!
  - Most are for launching and using enclaves
  - 2 new cryptographic operations
    - EREPORT – local attestation generation
      Local attestation = (prove an enclave is on the same machine)
    - EGETKEY – returns a key based on input parameters
      Allows for “sealing” = encrypting using an enclave specific key and saving outside the enclave for later use (on the same machine)

SGX Sealing

- Allows you to derive a key which is shared between all enclaves that are:

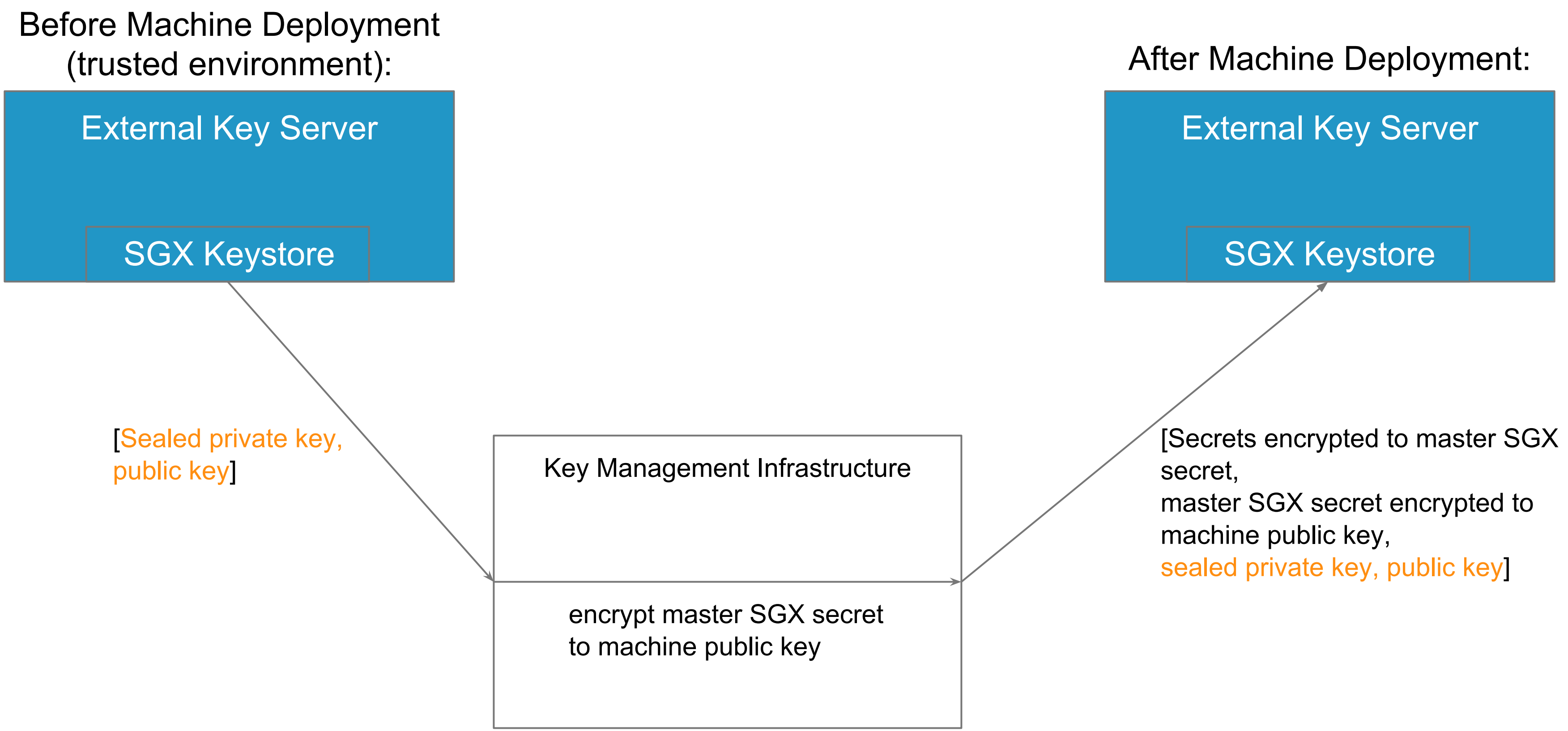
1. On the same system[1]
2. Signed by the same key[2]

- Key is persistent (through OS installs etc)

1. (same SEAL_FUSES)
2. (MRSIGNER = signing measurement) -- see See Figure 78 of “Intel SGX explained” https://eprint.iacr.org/2016/086.pdf

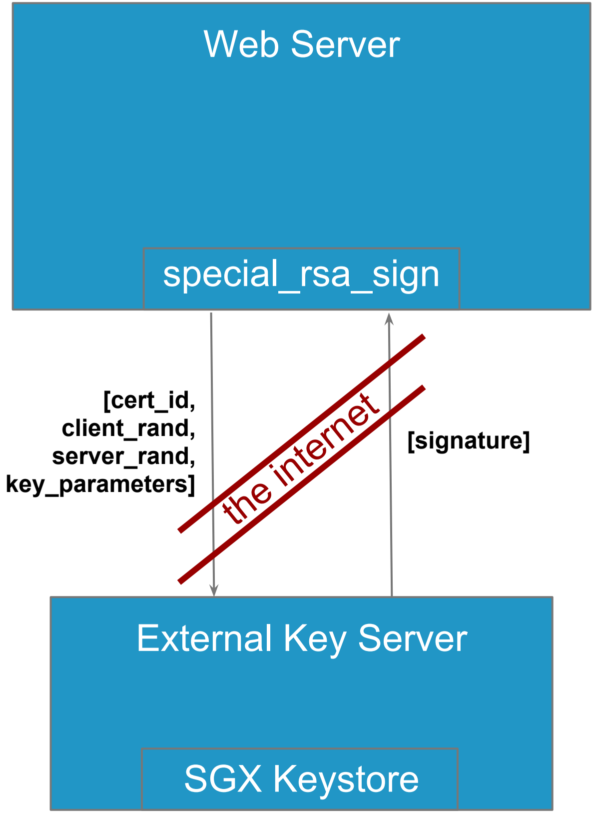
Seal via http://www.photolib.noaa.gov/htmls/anim0247.htm

Key Provisioning at Scale



Complexities with SGX

- Initial trust establishment
- Key rotation, key rotation, key rotation
- Multiple enclaves
- Secure handling of software signing keys
- Integration with SGXSDK & SGXSSL (OpenSSL in an enclave)
- SGX replay attack prevention



Questions?

serb@akamai.com
@erbbysam

Akamai

FASTER FORWARD