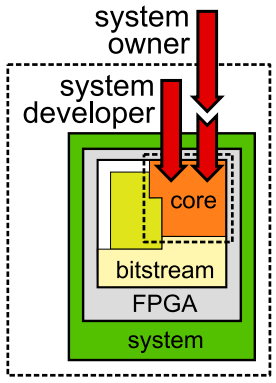
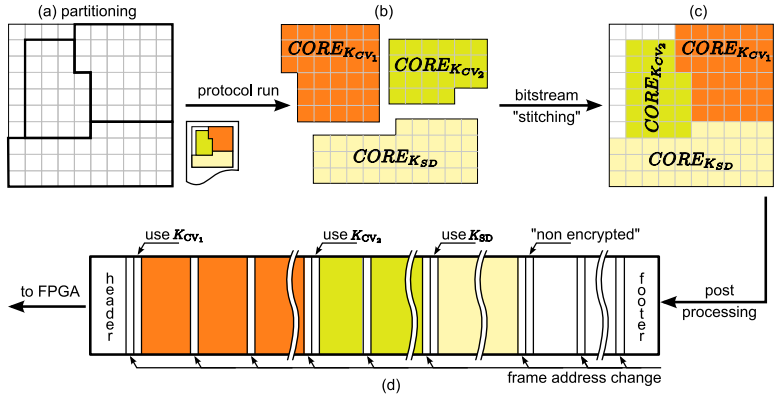
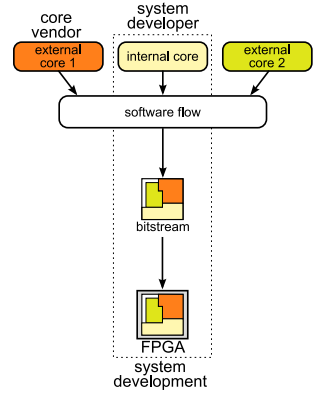
Securing SRAM FPGA designs – in distribution and in operation

Saar Drimer
www.cl.cam.ac.uk/~sd410







UNIVERSITY OF CAMBRIDGE
Computer Laboratory

This talk is about SRAM FPGA security issues, but mainly about distribution of designs

Outline:

- Use model – principals and how they interact
- A few attacks – reverse engineering, cloning, mislabeling, ...
- Cores and system protection
- A few solutions – encryption, authentication, ...
- Protecting many cores in a single FPGA design

Material is from "SRAM FPGA design security – a survey" and a bit of new work

Why are we concerned with FPGA security?

- **Size and complexity**: FPGA designs require great investment, so
- system developers want ready-made cores, but
- core vendors want to protect their designs and control their distribution.
- **Application space**: security attributes need to be investigated

Xilinx XC3020

Since 2000, FPGAs have ever more embedded functions that previously required external devices; designing for modern FPGAs requires specialization

Principals – **FPGA vendors** are risk averse, yet aggressive

- Introduce a new family every 12-18 months
- Use aggressive architecture and technology to compete with ASICs/ASSPs (and each other)
- Have an interest in helping their customers secure designs
- However, they will only introduce features that make financial sense

Design protection solutions require adoption by the FPGA vendors; they must therefor be beneficial to their bottom line

Principals – two types of **system developers** concerned with security

Cost-conscious:

- Achieve design goals at lowest cost; security is secondary
- Short term protection (months to a few years)
- Generally trusting (can't afford not to be)

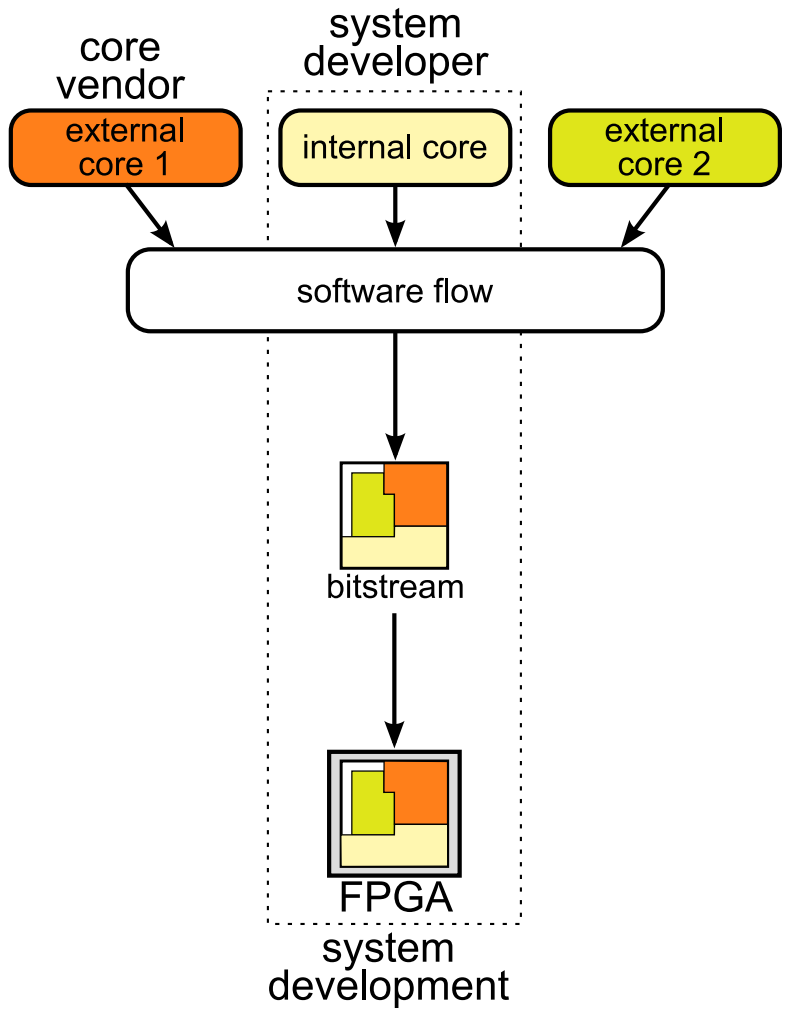
Security-conscious:

- Security is a priority; willing to pay for it
- Long term protection (years to decades)
- Distrusting: require proven primitives, rigorous audit of supply chain and verification of design tools

In general, when we speak of system developers we think of cost-conscious ones; no single solution will satisfy both

Principals – **core vendors** sell ready-made digital functions

- Sell designs in HDL or netlist form
- Currently, forced to use blanket licensing arrangements
- Cannot enforce licensing terms once core is handed off
- Common practice in industry is to rely on risk perception and agreements/contracts ("social deterrents")



Principals – the rest

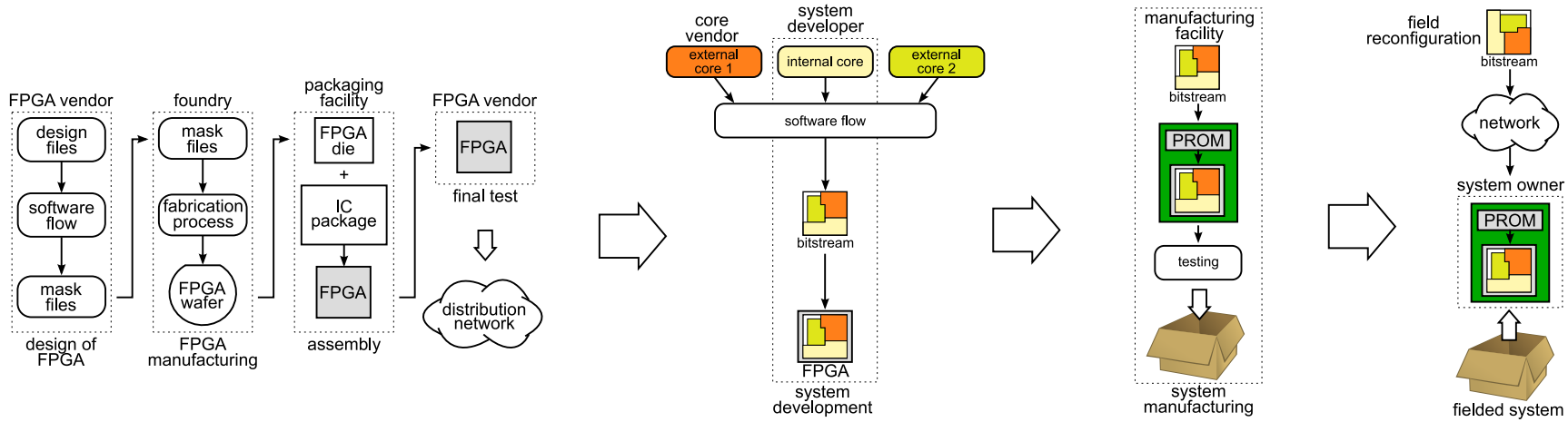
- **EDA vendor** – software tools
- **Foundry** – fabricates the FPGA
- **System manufacturer** – manufactures the product
- **System owner** – whoever ends up using the system in the field

By necessity, some of these principals are considered trusted parties

Of course, we shouldn't forget

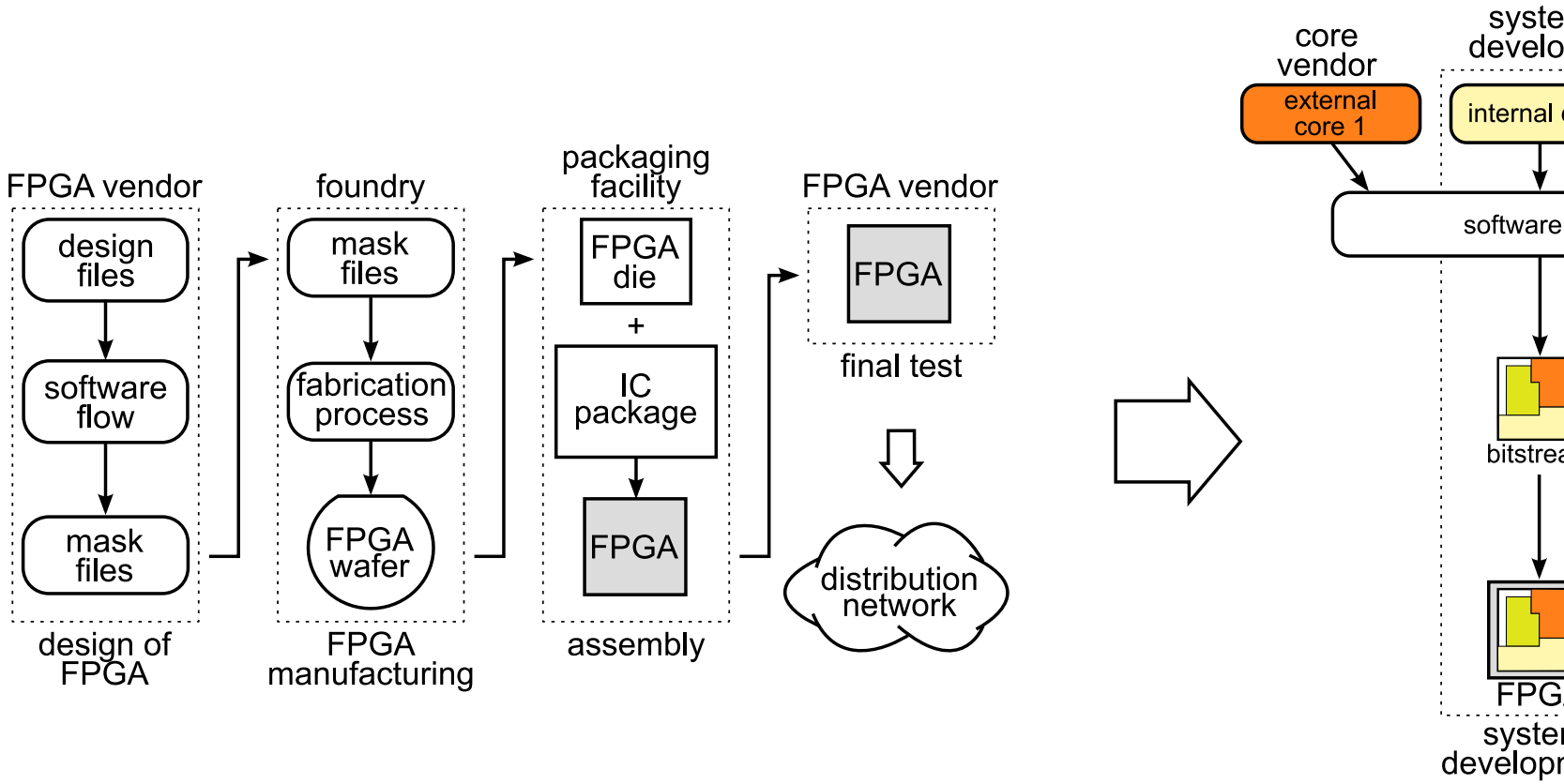
- **Academics** and **hobbyists**

Understanding the use model of FPGAs is a key to evaluating its security properties



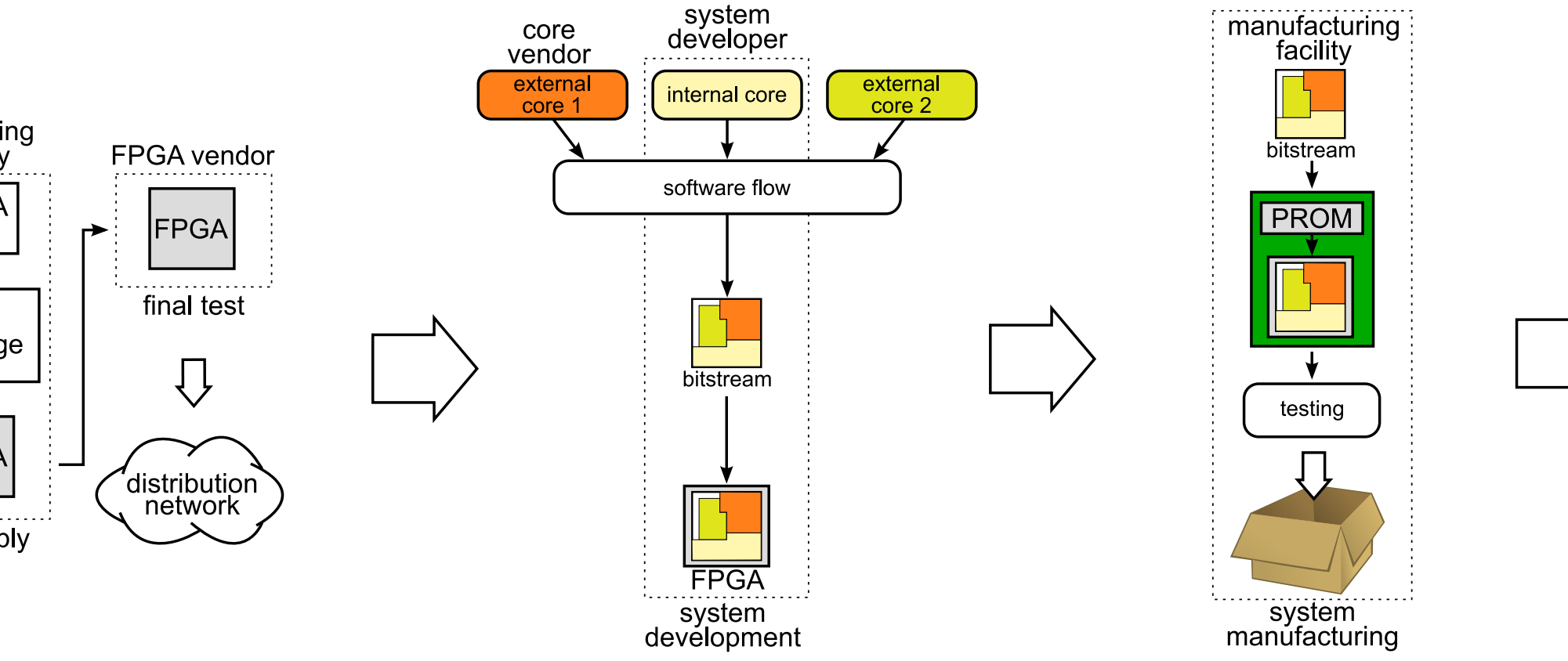
From birth to bin – the life-cycle of an FPGA and FPGA system

Use model – an **FPGA** is born



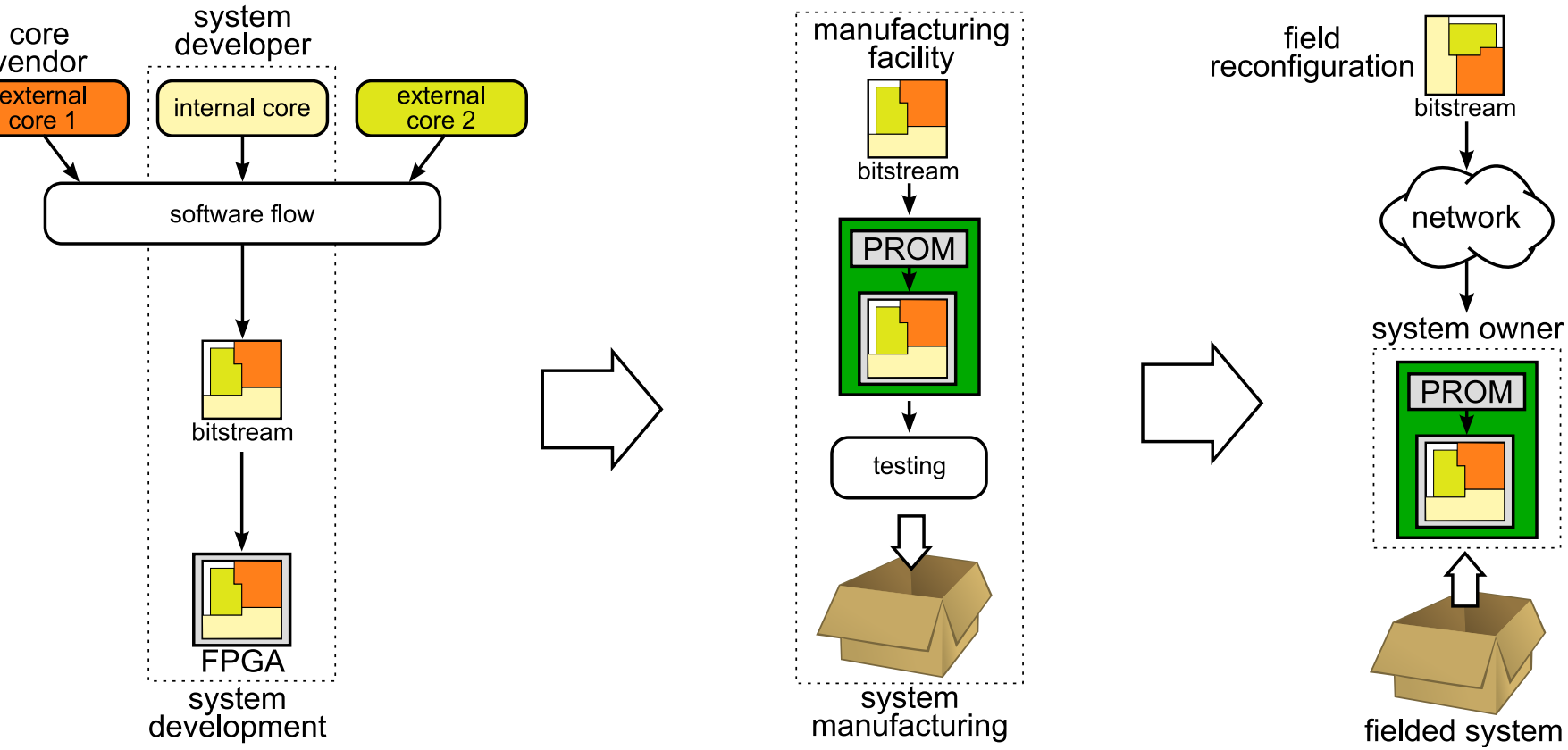
FPGA design and manufacturing process is like an ASIC's, except that the result is a commodity reprogrammable device; many entities are involved in the process

Use model – an **FPGA system** is born and educated



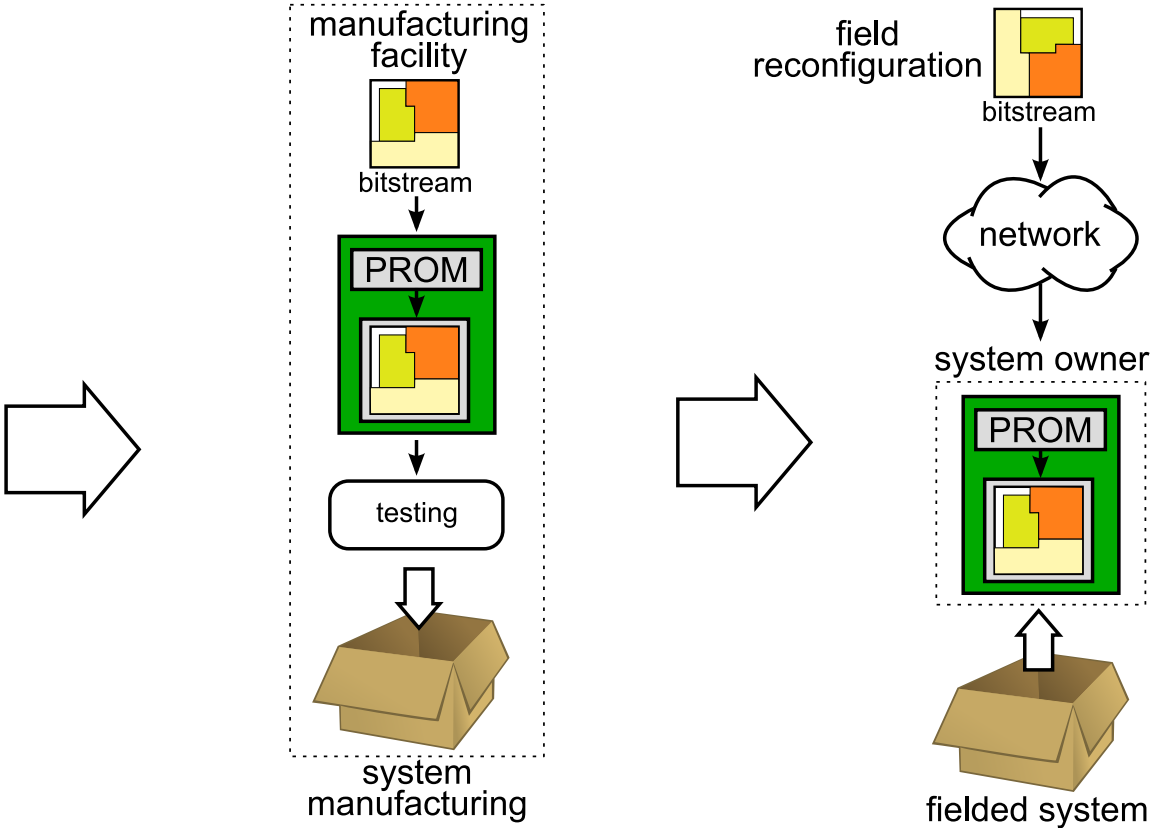
Cores from different sources are combined into a single description of a digital function, which is then distilled into a "bitstream" used to "program" the FPGA

Use model – graduation



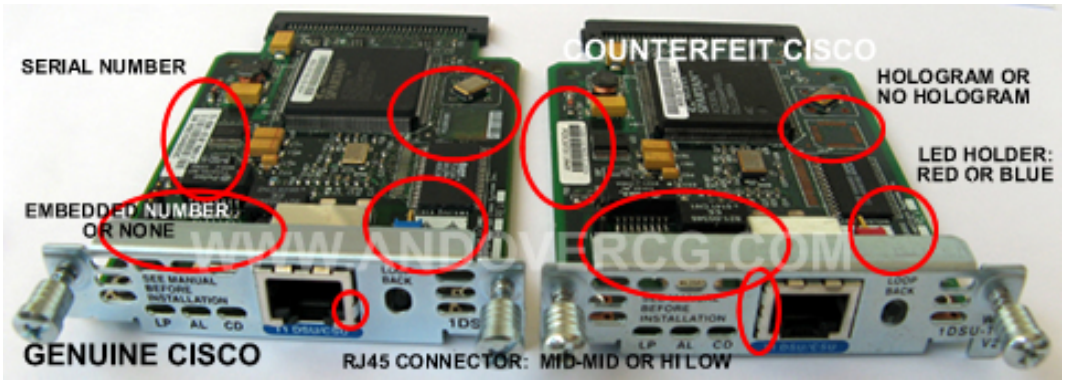
The system is manufactured, tested, packaged, and sent for distribution

Use model – leaves the nest, but stays in touch



The system owner ("enemy") has possession of the system, though the developers can update the FPGA's functions through "field reconfiguration"

Design theft – a real problem

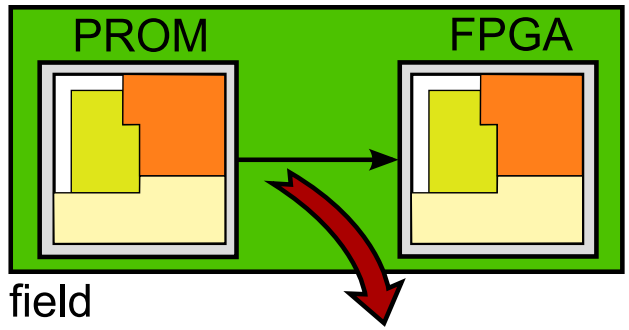


Cisco vs. "Chisco"

- "10 percent of all high tech products sold globally are counterfeit"
- "Chisco" – recently, US and Canada located/seized $76m worth of counterfeit Cisco gear sourced from China; security implications can be quite severe
- Cloning is happening for popular consumer products (network equipment, TVs, etc.)

[Alliance for Grey Market and Counterfeit Abatement; FBI/DOJ]

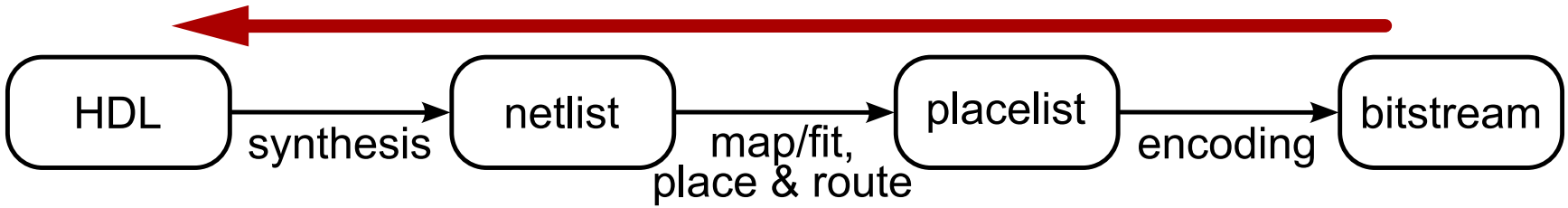
Cloning, overbuilding and mislabeling



- **Cloning** – easy attack, considered most common security vulnerability of SRAM FPGAs
- **Overbuilding fraud** – also easy, simply make more of the product, sell ones that did not pass testing, or over-report failed system
- **Mislabeling** – modifying the marking on the package; buyer doesn't know until he programs the FPGA
  - Speed grade mislabeling are especially hard to detect
  - Maybe an open suite of test vectors can help?
  - Easiest solution is to buy from an authorized distributor (not eBay)

Reverse engineering the bitstream

The transformation of an encoded bitstream into a functionally equivalent description of the original design



Easy: extracting content from BRAMs and LUTs

- "ULogic FPGA netlist recovery" (they mean "placelist")

Hard: inferring complete functionality for reuse

Partial reversal can reveal information as well

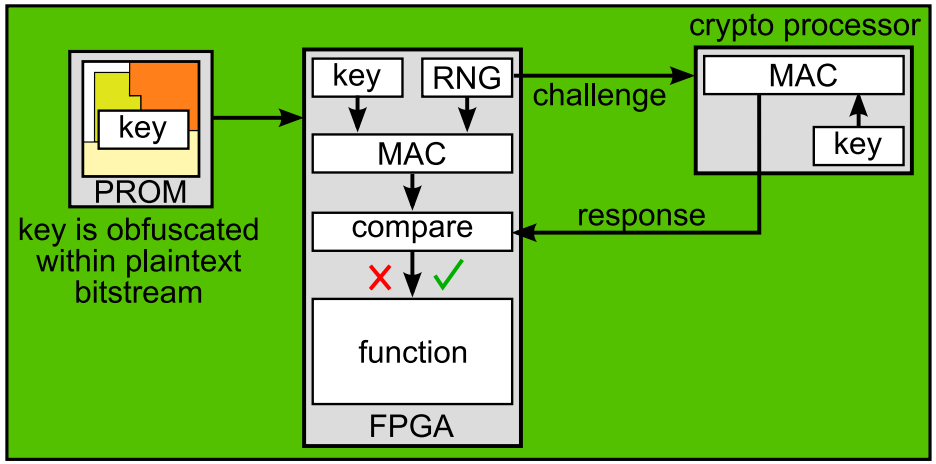
Reverse engineering – what shall we do?

Given the high value of designs, incentives are right for attackers to invest more in reverse engineering;

therefore, we should start moving away from design protection schemes that rely on bitstream obscurity

- JBits API
- Open-source bitstream format (XC6200, Atmel FPSLIC)

Design theft deterrents – rely on obscurity



The idea is to increase the difficulty bar of cloning

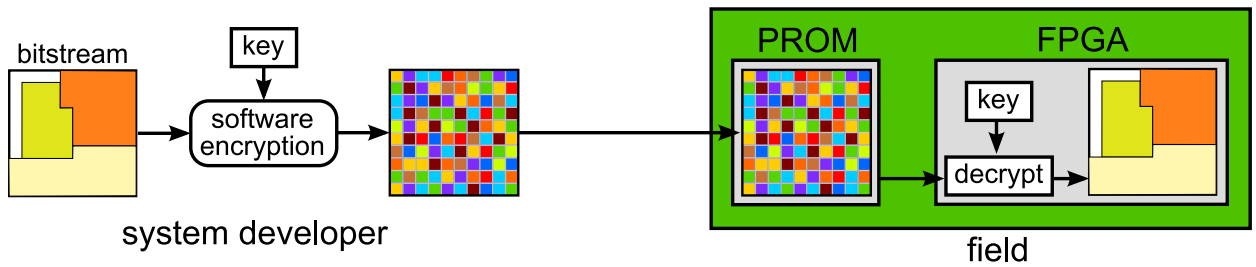
- First proposed by Kessner in 2000, now both Altera and Xilinx offer similar reference designs

Through a challenge-response exchange with a non-volatile crypto device on the same board, the bitstream can "authenticate" that it is operating in the correct environment

Possibly working for now, but for how much longer?

Bitstream encryption – a good start

Now standard in high-end FPGAs: the bitstream is encrypted by the software and decrypted by the FPGA using a user-defined key



Encryption only provides confidentiality while distributing bitstreams, only protects whole bitstreams, and is not available for low-end FPGAs

Encrypted bitstreams can still be manipulated

Bitstream authentication – a good idea

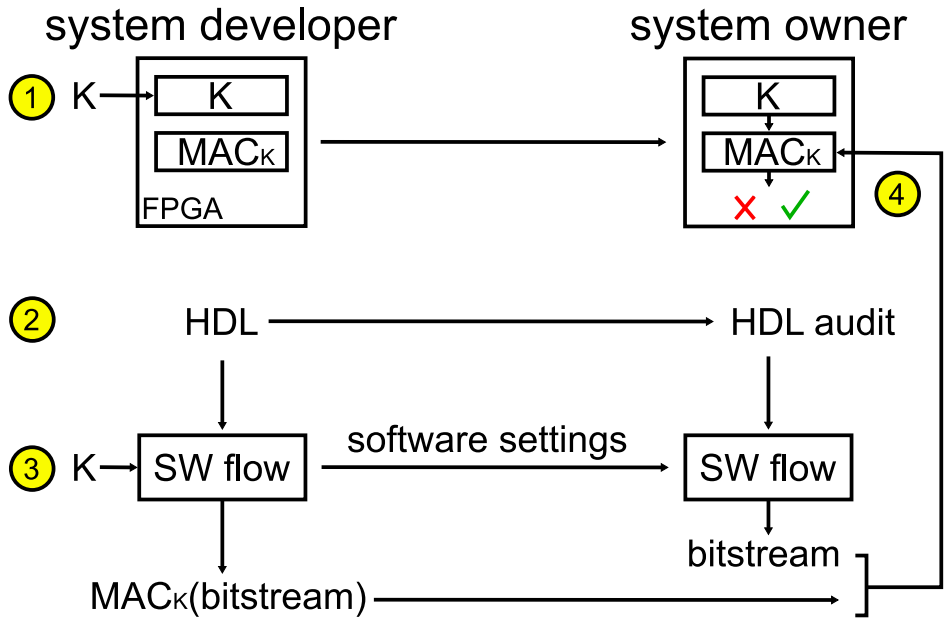
If encryption protects the bitstream in distribution, authentication protects the correct and intended operation of the FPGA

Authentication provides entity identification and cryptographic data integrity

- Since the FPGA doesn't retain state, replay of old versions of the bitstream is still a problem
- Relay attacks are possible as well; distance fraud can be an issue for security applications
- Can provide role- and identity-based access control

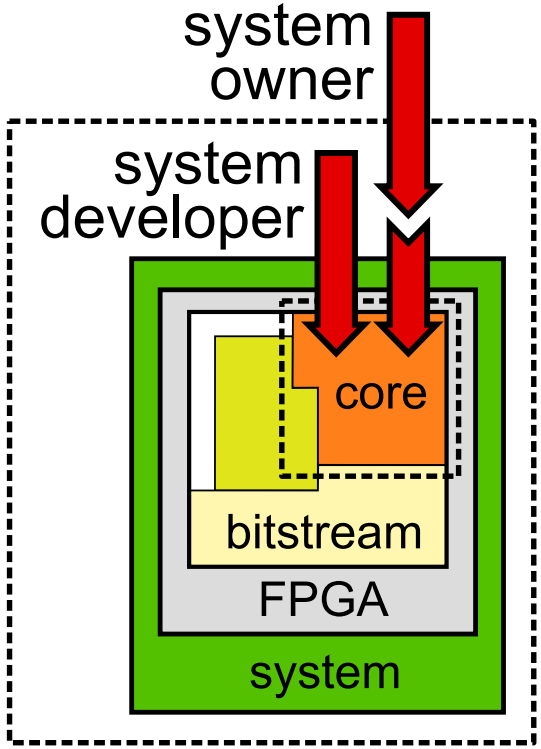
Without extra devices (and tamper proofing), preventing denial-of-service attacks is very difficult

Authenticating without encryption can allow code audit – a voting machine example



1. System developer programs authentication key into FPGA and sends system to voting authority
2. Developer send HDL to authority for audit
3. Both run the code through the software flow under exact conditions
4. Authority programs the FPGA with the locally generated bitstream with is authenticated using the MAC from the developer

Two types of design protection



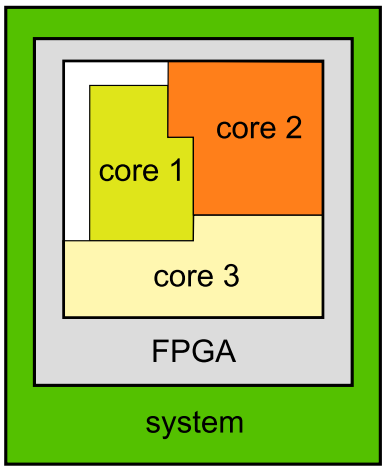
System protection:
protects designs from malicious users

Core protection:
protects designs from malicious users
and system developers

We already have "system protection" for some FPGAs; how can we efficiently provide "core protection"?

What are we actually after?

An ideal distribution model will allow a system developer to evaluate, simulate, and integrate cores from multiple vendors while 1) core vendor is able to restrict each instance to a particular FPGA and 2) the cores' confidentiality and authenticity are assured



The scheme should be easy to use and deploy, cheap, and relatively transparent to users; can we do it?

Most solutions to date deal with the vASSP problem or "system protection"

ASSP: application specific standard product – a commodity ASIC that performs a fixed set of functions (USB controller, MPEG encoder, etc.)

Virtual ASSP: a single third-party core that occupies the entire FPGA without contribution from the system developer

vASSP and compiled code distribution is only a small subset of the "business"

- Kean, Bossuet et al., suggest "system protection" schemes
- Simpson and Schamont, Guajardo et al. deal with secure microprocessor code distribution using PUFs
- Güneysu et al. uses the user logic and asymmetric crypto for key establishment

However, the vASSP scenario (and processor code dist.) is...

- only a small part of the "core distribution business", and
- **already reasonably solvable by using bitstream encryption and having the core vendor sell the FPGA to the developer complete with an encrypted bitstream**

We start with the vASSP scheme of Güneysu et al.

Key ideas:

- Maximize the use of the user logic
- Minimize additions to the configuration logic
- Use asymmetric key establishment between core vendor and FPGA
- Use a "personalization bitstream" encrypted under the FPGA vendor's symmetric key

Key Derivation Function using asymmetric cryptography:

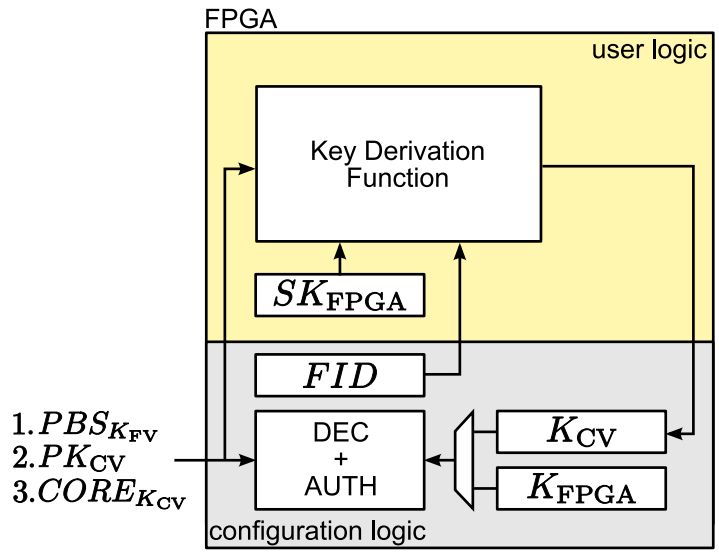
$$K_{\mathrm{AB}} = \mathrm{KDF}(\mathit{SecretK}_{\mathrm{A}}, \mathit{PublicK}_{\mathrm{B}}, \mathrm{data}) = \mathrm{KDF}(\mathit{SecretK}_{\mathrm{B}}, \mathit{PublicK}_{\mathrm{A}}, \mathrm{data})$$

[Güneysu, Möller, Paar: "Dynamic intellectual property protection for reconfigurable devices"]

Assumptions and requirements

Assumptions:

- Each FPGA has a unique non-secret "FPGA ID" (FID)
- Bitstreams are authenticated and encrypted
- PKI is used for certifying public keys



FPGA vendor:

- Generates symmetric key K_{FPGA} and asymmetric pair SK_{FPGA} and PK_{FPGA} (for each individual or group of FPGAs)
- Creates "personalization bitstream" that contains SK_{FPGA}
- Embeds key K_{FPGA} into FPGA in an OTP keystore

vASSP protection protocol

A. SETUP:

FPGA vendor		**system developer**
$E_{K_{FPGA}}(\mathrm{PB})$	$\xrightarrow{FID, PK_{\mathrm{FPGA}}, \mathrm{PB}_{K_{\mathrm{FPGA}}}}$	

FPGA vendor encrypts the personalization bitstream (PB) with K_{FPGA}; data is sent with the FPGAs

vASSP protection protocol

B. LICENSING:

core vendor		**system developer**
	$\underleftarrow{FID, PK_{\mathrm{FPGA}}}$	
$K_{\mathrm{CV}} = \mathrm{KDF}(SK_{\mathrm{CV}}, PK_{\mathrm{FPGA}}, FID)$		
$E_{K_{\mathrm{CV}}}(\mathrm{CORE})$		
	$\underrightarrow{PK_{\mathrm{CV}}, \mathrm{CORE}_{K_{\mathrm{CV}}}}$	

Both core vendor and system developer need to verify the certificates of public keys in order to prevent decryption of cores or a "Trojan core"

vASSP protection protocol

C. PERSONALIZATION:

system developer		**FPGA**
	$\xrightarrow{\mathrm{PB}_{K_{\mathrm{FPGA}}}}$	$E^{-1}(\mathrm{PB}_{K_{\mathrm{FPGA}}})$
	$\xrightarrow{PK_{\mathrm{CV}}}$	$K_{\mathrm{CV}} = \mathrm{KDF}(SK_{\mathrm{FPGA}}, PK_{\mathrm{CV}}, FID)$

The FPGA generates K_{CV} using the personalization bitstream and stores it in a designated key-store

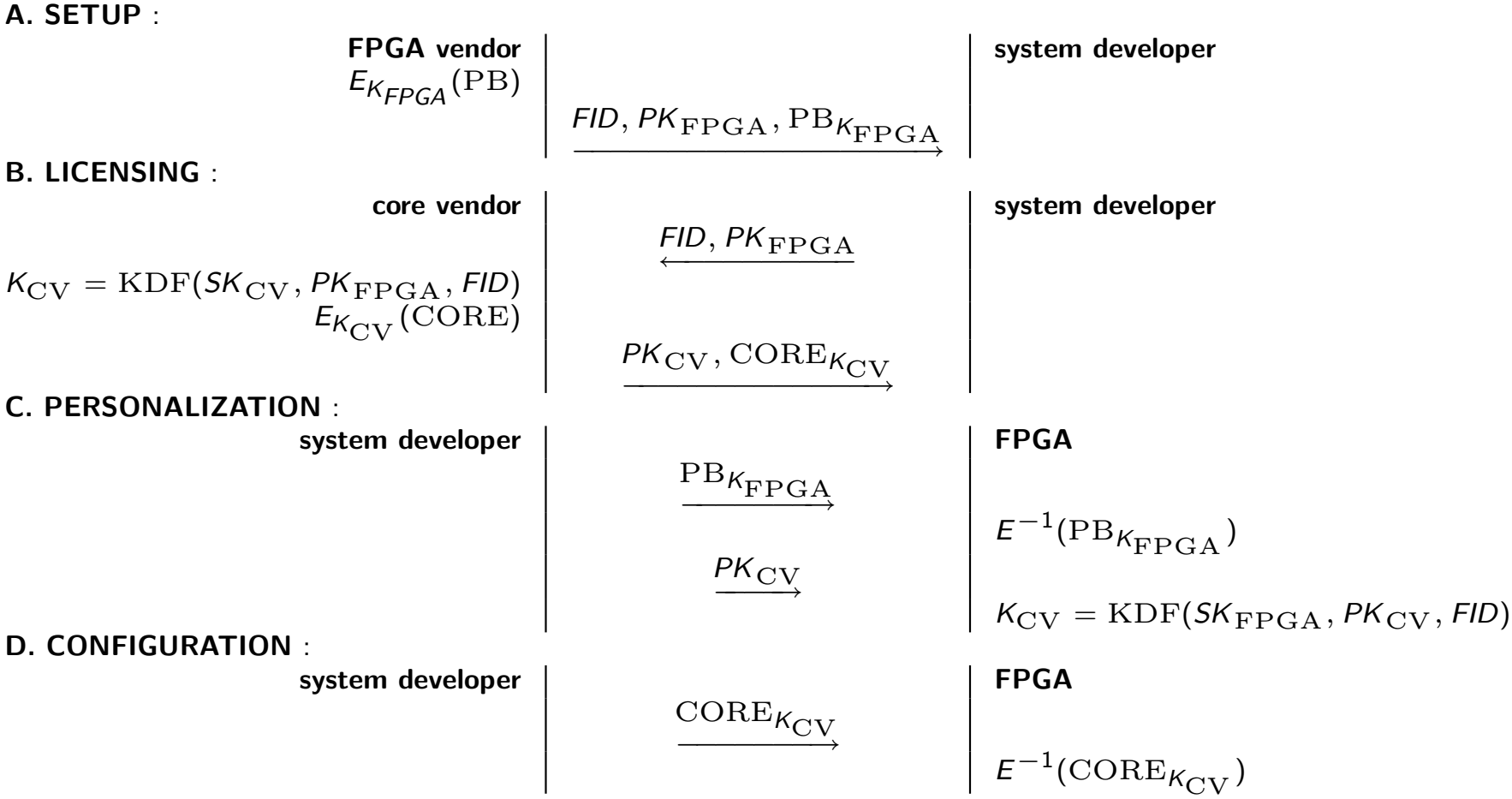
vASSP protection protocol

D. CONFIGURATION:

system developer		**FPGA**
	$\xrightarrow{\mathrm{CORE}_{K_{\mathrm{CV}}}}$	$E^{-1}(\mathrm{CORE}_{K_{\mathrm{CV}}})$

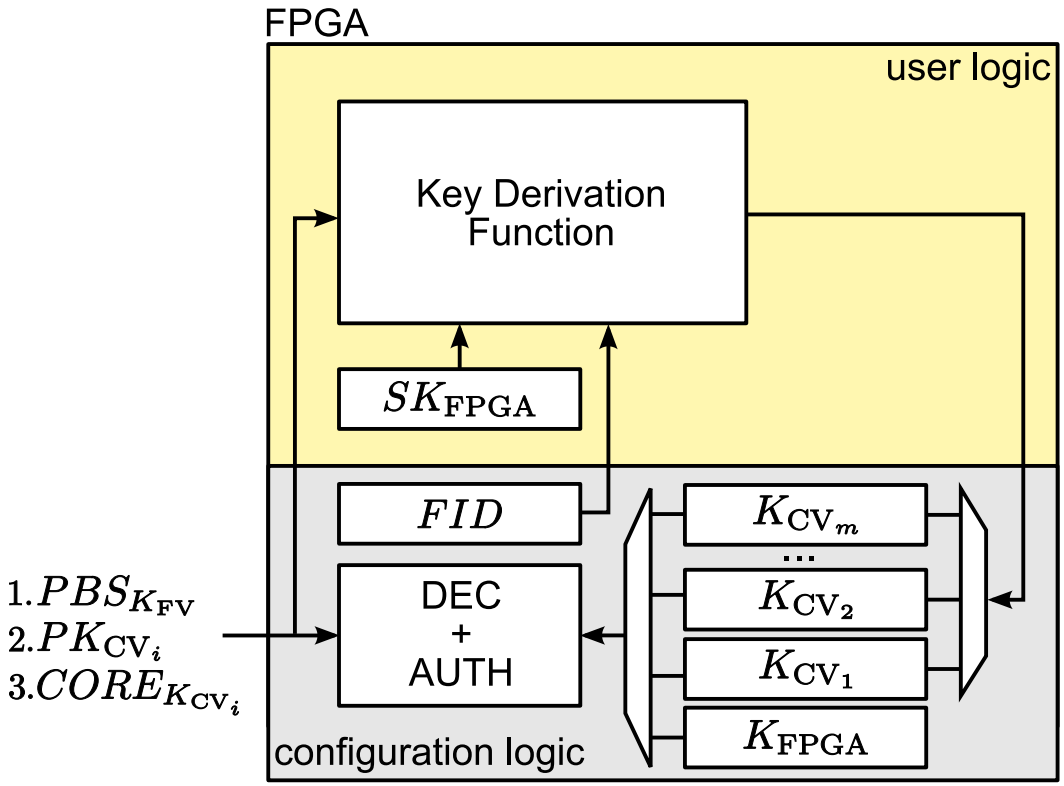
The core is decrypted using K_{CV}

vASSP protection protocol



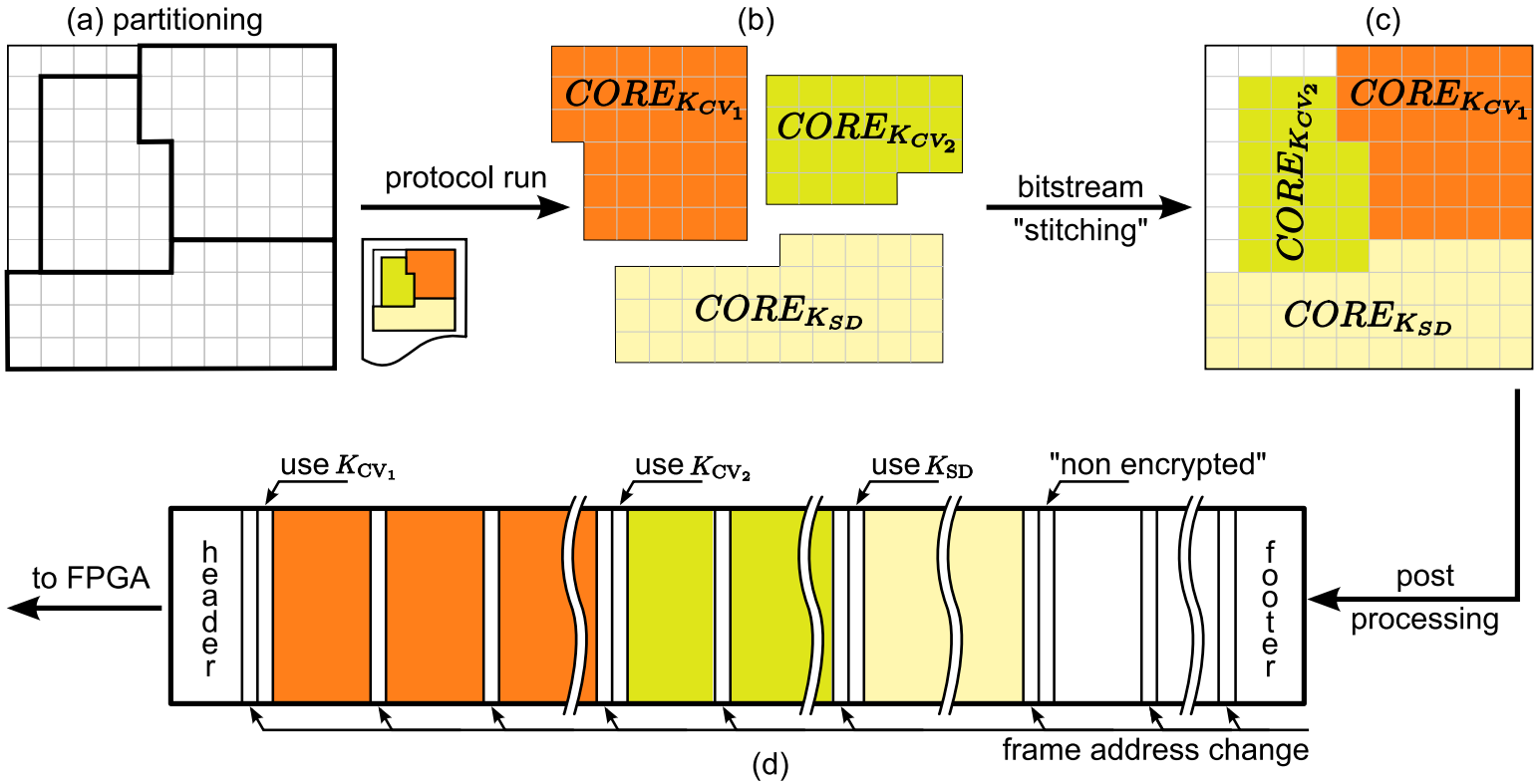
Notice that the system developer only relays messages

Extension of the scheme – first, we will need more key storage



The amount of key storage corresponds to how many cores we can protect in a single bitstream

Extension – changes to the design flow and bitstreams



By extending the vASSP solution, and using existing software tools and small changes to the bitstream format, we could protect multiple cores from multiple sources

Evaluation of the many-core protection scheme

Advantages:

- Small, scalable and optional to developer (opt-in; otherwise ignore)
- Uses established primitives (no PUFs, RNGs required)
- Software is mostly there already (modular design, partial recon.)
- Configuration times are not affected much (unless volatile key storage is used)
- Incentives are well aligned

Evaluation of the many-core protection scheme

Problems:

- Loss of optimization compared to HDL-level integration
- Implementation must be fault-tolerant and tamper resistant
- Limited simulation (use a crippled-version)
- System developer needs to trust the core vendor not to have malicious code in the core
- Bandwidth may be an issue for large design exchanges

Conclusion: lots of work ahead!

We've covered only a small subset of the topic; lots more in the survey

Latest version of survey:
`http://www.cl.cam.ac.uk/~sd410/papers/fpga_security.pdf`

FPGA Design Security Bibliography:
`http://www.cl.cam.ac.uk/~sd410/fpgasec/`

My work and contact info:
`http://www.cl.cam.ac.uk/~sd410/`

[Picture: Sebastien D'ARCO, Creative Commons Attribution ShareAlike 2.5 License]

Thanks!