HDCP2

von den Schwierigkeiten, ein Security-Protokoll zu entwerfen

OpenRheinRuhr
9. November 2013

Martin Kaiser

What?

- designing a security protocol is difficult
- “protocols live forever”
- open-source software is very useful for analyzing closed systems

Overview

- HDCP2
- Building blocks
- Authentication
- AKE protocol
- Problems with the AKE protocol
- Pairing, pairing data
- Breaking the AKE protocol
- Doing this on real devices
- Recap: What did we achieve?
- Summary

About me

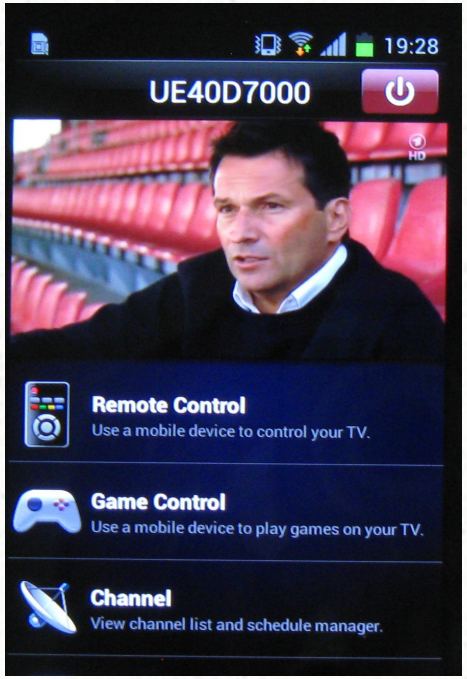
- writing embedded software for Digital TVs
- involved in creating the CI+ Pay-TV standard
- Wireshark Core Developer
- http://www.kaiser.cx

HDCP2

- High-bandwidth Digital Content Protection, version 2
- secure transmission of premium Audio-Video content
  - one part of a DRM system
- HDCP2 != HDCP

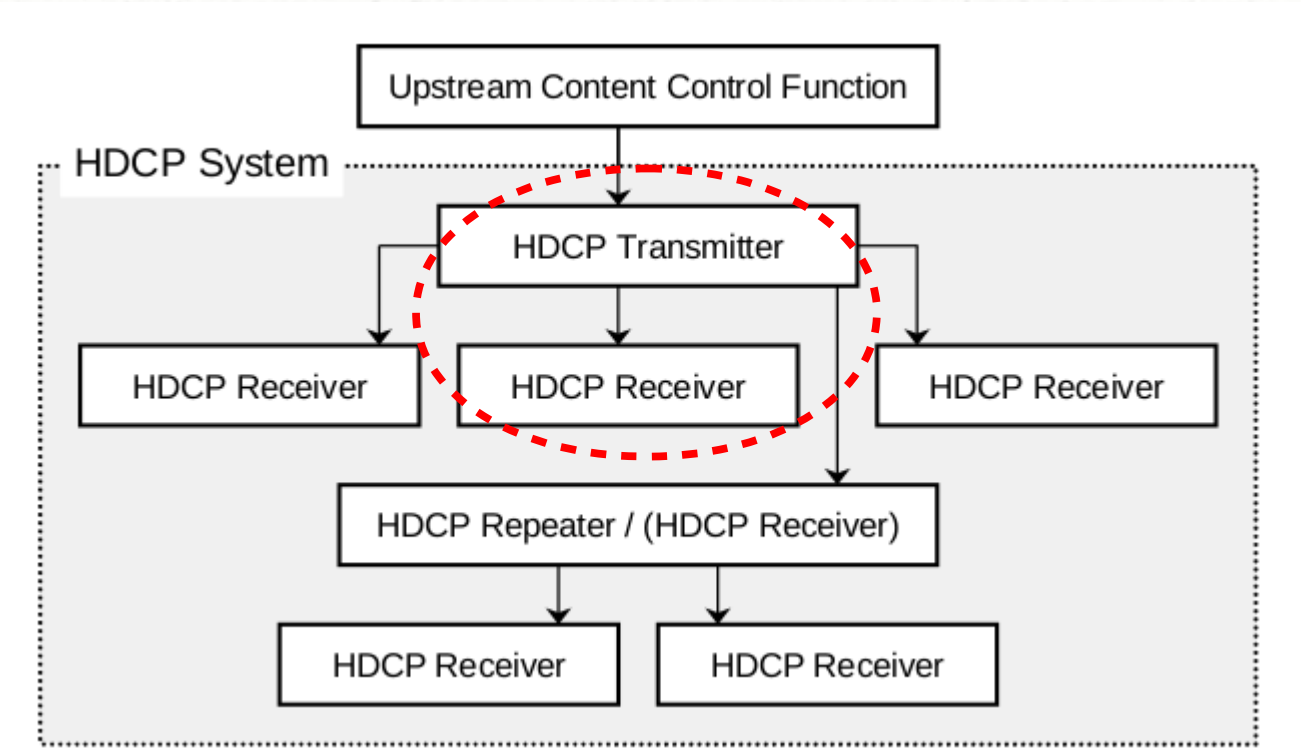
HDCP2 examples

- streaming of TV content to a smartphone
  - remote control app
- Miracast
  - mirror the smartphone screen on the TV



Miracast

Basic concepts



typical scenario: one transmitter, one receiver

Building blocks

- authentication
- renewability (revocation)
- transmission of payload data

HDCP2 authentication

- AKE (Authentication and Key Exchange)
  - result: master key k_m
- locality check
- session key exchange
  - result: session key k_s

AKE protocol

- transmitter verifies the receiver's identity (i.e. the receiver's certificate)
- transmitter creates the master key k_m
- transmitter encrypts k_m (using RSA) and sends it to the receiver
- check that both parties have the same k_m
- make sure that k_m can be reused

AKE protocol in detail

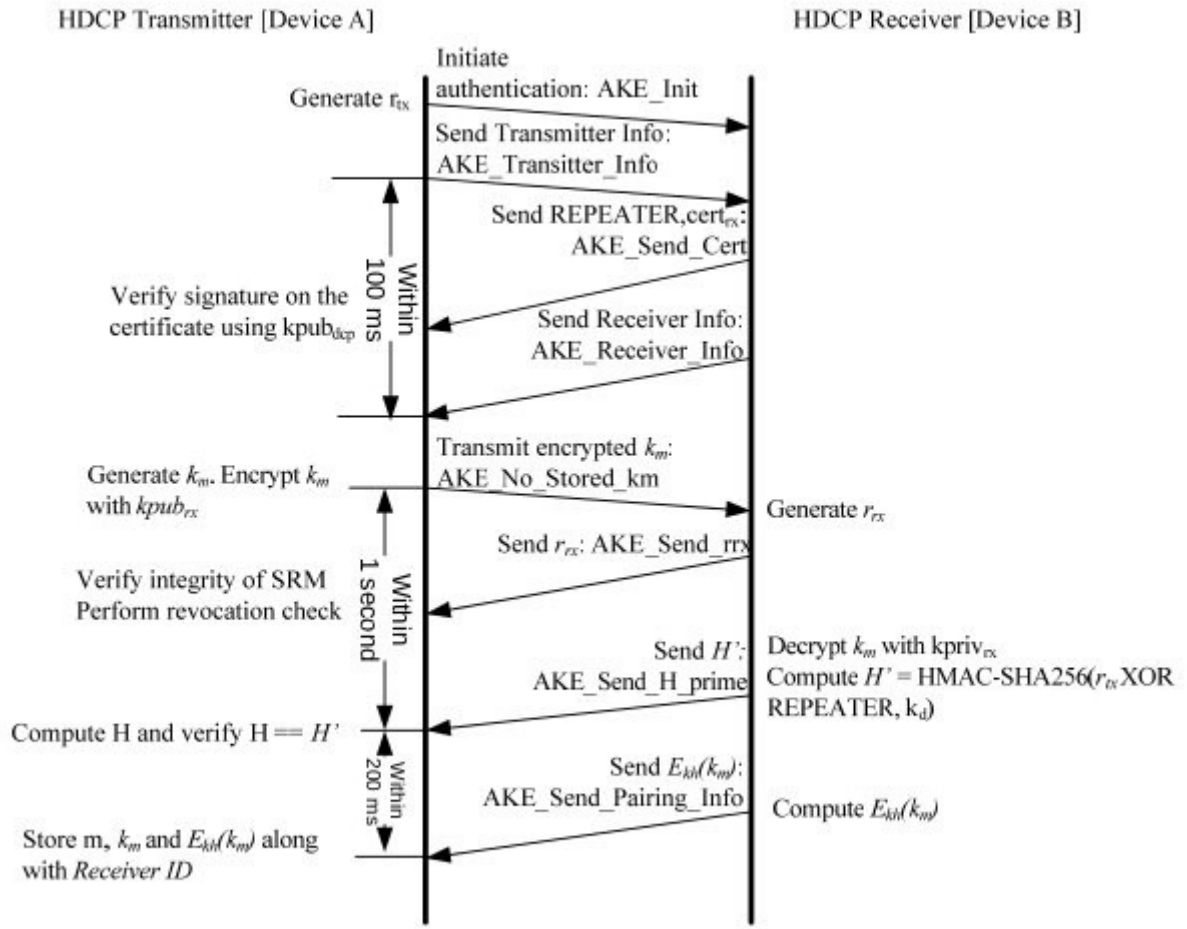


Figure 2.1. Authentication and Key Exchange (Without Stored k_m)

What's wrong with AKE?

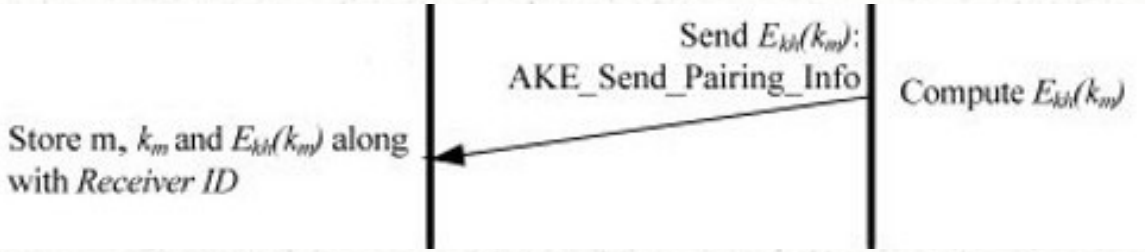
- Nothing, but ...
  - transmitter is not authenticated
  - messages are not signed
  - receiver sends a random number - but it's not used in subsequent calculations

Pairing

- RSA calculation is time-consuming
- reuse the master key between the same two devices
  - both devices must store the master key
  - assumption: receiver can't do this
- the transmitter stores the receiver's copy of the master key
  - receiver encrypts the master key with an internal secret key
  - transmitter stores the encrypted version

Pairing data

receiver encrypts the master key k_m for storing by the transmitter

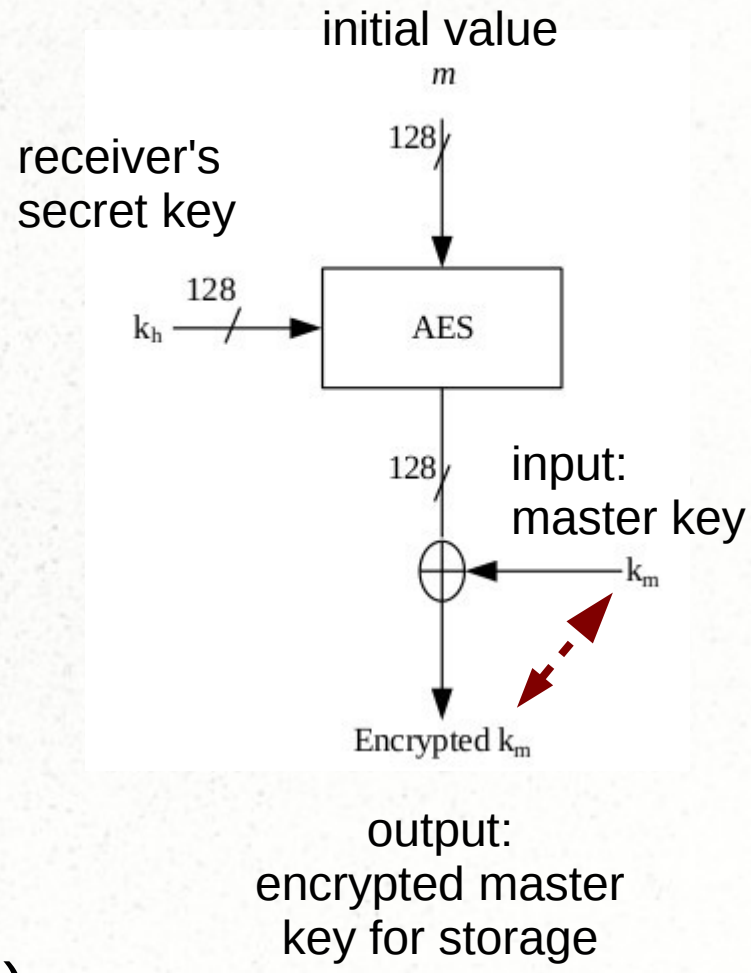


transmitter stores pairing data

- encrypted k_m (from the pairing message)
- clear k_m
- initial value m (transmitter's random number)
- receiver ID

Receiver generates the pairing data

- receiver encrypts the master key for storage
- encryption and decryption are the same operation
  - doing this twice gives you back the data in the clear
- don't ever reuse the initial value m
  - but this is based only on data from the transmitter ;-)



Attacking the AKE protocol (I)

- capture an HDCP2 session including the AKE
  - capture the initial value m
  - capture encrypted k_m
  - capture the verification value H'

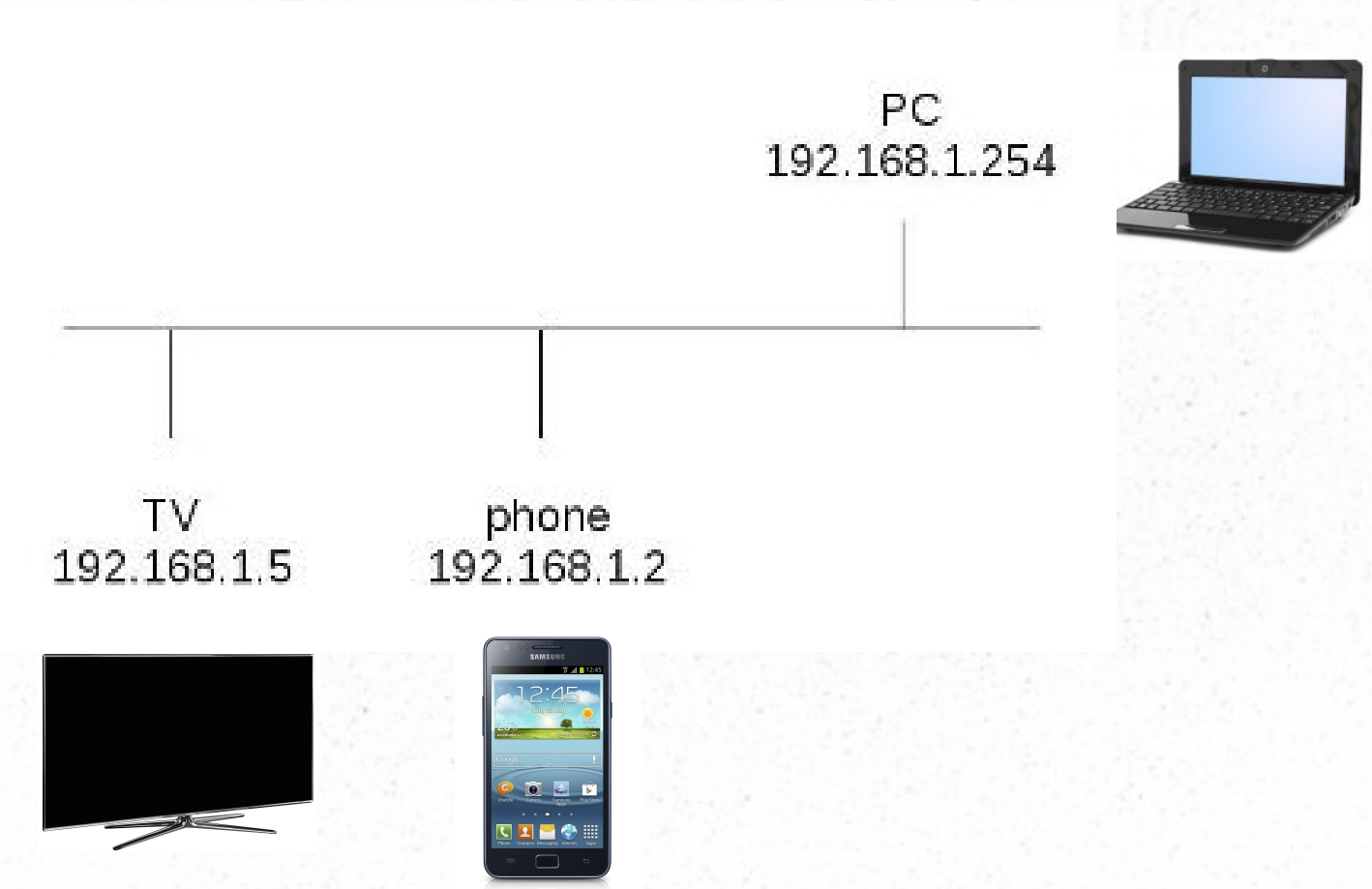
Attacking the AKE protocol (II)

- run the AKE protocol as a transmitter
- receiver performs the calculation of pairing data
  - trick it into using forged values
  - the same initial value m as in the captured session
  - encrypted k_m instead of clear k_m
- the receiver does not create paring data, it decrypts the pairing data from the captured session
  - this recovers the clear master key k_m

Setup

- Samsung TV, Galaxy S2 phone
  - TV is streaming to the phone's remote control application (*dual view*)
  - TV is the transmitter, phone is the receiver
- HDCP2 does not specify how transmitter and receiver find each other
  - Samsung uses DLNA
  - we don't implement this, we just add our fake transmitter to the network and start sending...

Test network



Capture the AKE protocol

- HDCP2 AKE messages are simple
  - always in the clear
  - no context required for parsing them
- Wireshark HDCP2 dissector
  - on top of TCP
  - heuristic dissector, no well-known TCP port
  - available in Wireshark >= 1.8

Fake AKE protocol

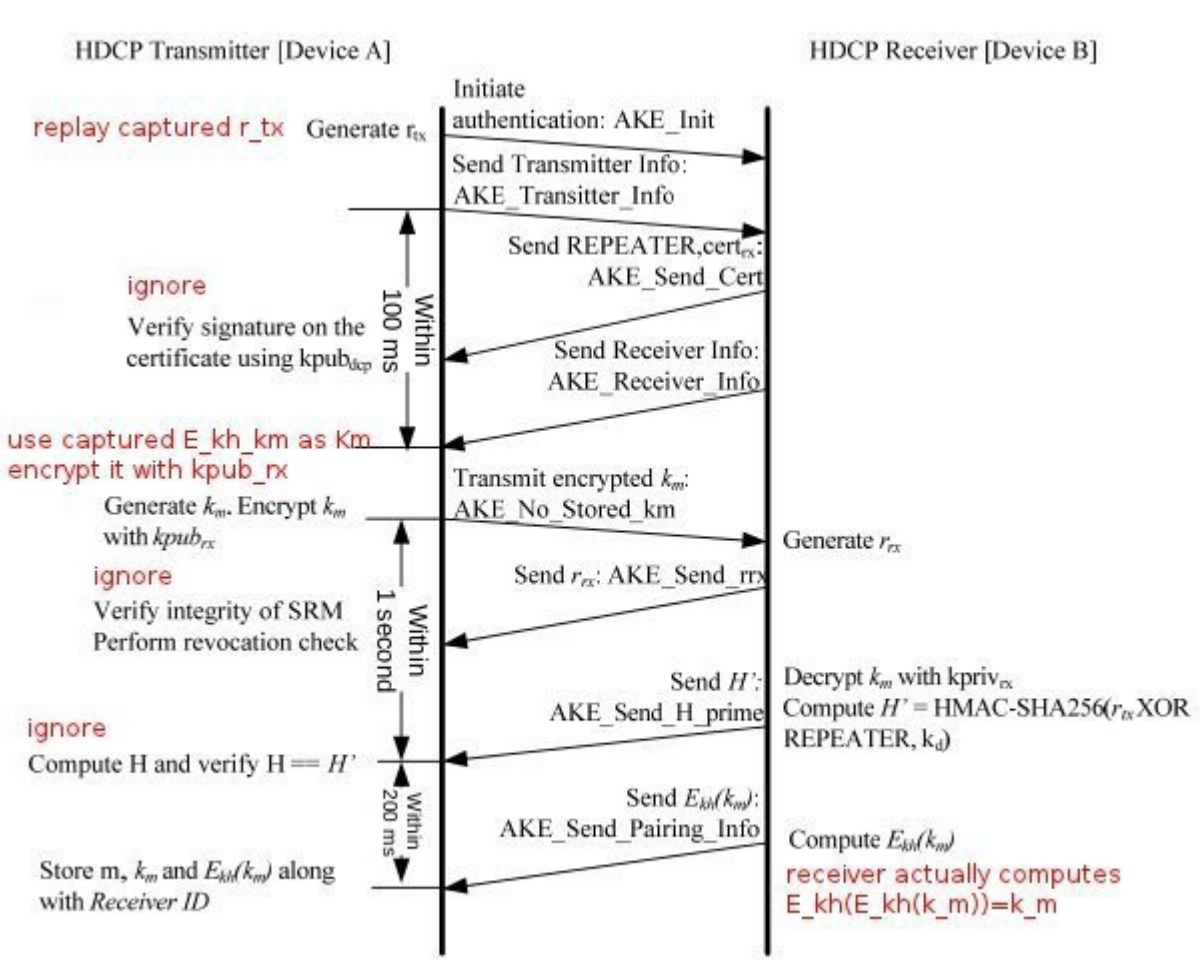


Figure 2.1. Authentication and Key Exchange (Without Stored k_m)

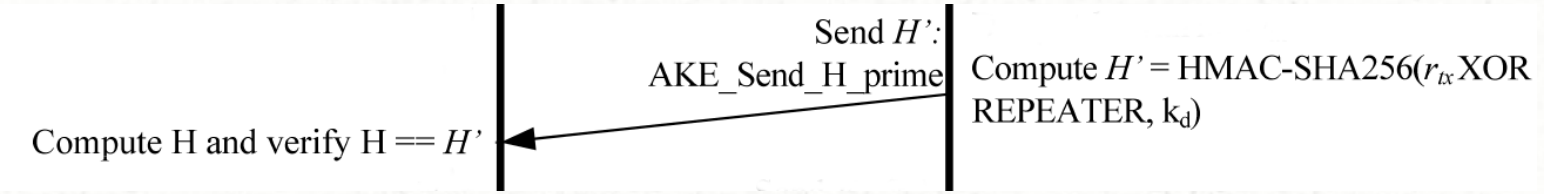
RSA encryption

- normally, the transmitter creates the master key k_m and encrypts it using RSA
  - we encrypt the AES-encrypted k_m from the pairing data
- RSA encryption
  - RSASSA OAEP with SHA256, MGF1 with SHA256
  - not supported by OpenSSL at the time I tried this first
  - libgcrypt can do this

Run the fake AKE protocol

- phone will only listen on the HDCP2 TCP port after successful DLNA discovery
  - we let the phone and TV do the DLNA part
  - phone does not restrict TCP connections to the TV's address ;-)
  - transmitter may initiate the AKE protocol at any time
- all messages can be pre-computed
  - none of our fake transmitter messages depends on a previous (variable) answer from the receiver
- use Wireshark to parse the receiver's answers

We have the master key! Really?



- receiver calculates a verification value H' and sends it to the transmitter
  - H' depends on the master key k_m
  - we captured the H' of the original AKE protocol run
- transmitter calculates the same value and compares them
  - can we calculate H==H' for the original AKE run?
- this needs HMAC-SHA256 and AES-CTR
  - all supported by OpenSSL

Recap

- we can get the master key of a captured HDCP2 authentication
  - we just need to speak to the receiver for a moment
  - the master key will be the same for all past and future sessions between the two devices
- this is a protocol weakness, it does not require a buggy implementation in one of the devices
- it's not enough for decrypting the AV content
  - license constant lc_{128} is missing

Fixing this

- HDCP 2.2
- change the formula for the initial value
  - $m = r_{tx} \| r_{rx}$
  - the inital value depends on random numbers of both transmitter and receiver
  - a fake transmitter cannot force the same initial value m
- an HDCP 2.2 device does not do pairing with an HDCP 2.1 device

Summary

- Protocols are complicated
  - if possible, don't define your own protocol
- understand the limitations of cryptographic primitives you're using
- even for the closest of systems, open-source software helps to analyze and understand them
- adding *your* protocol to Wireshark is easy
  - see my next talk ;-)

Thank you for your attention.

Questions?