1. The lifecycle of an exam question in a fictitious university includes at least the following stages, which take place over several months:

   (i) Professor invents question.

   (ii) Chief examiner sanity-checks it.

   (iii) Professor amends it if necessary.

   (iv) External auditor sanity-checks it.

   (v) Professor amends it again if necessary.

   (vi) Chief examiner approves final version.

   (vii) Clerk prints question in required number of copies.

   Following a scandal whereby some dishonest candidates got hold of questions ahead of time, thus forcing the whole exam to be invalidated and repeated to the dismay of the honest participants, the university has put pressure on its departments to ensure this will not happen again.

   (a) The Head of Department A, where the leak occurred, is now paranoid about computer networks and insists that no exam question shall ever reside on any networked computer system until after the corresponding exam takes place.

      (i) Describe four ways that a determined undergraduate might nonetheless get hold of exam questions before the exam even if that requirement were observed.

Physical methods

If you're lucky, they'll show it to you while you're in their lecture.

Observation of someone who is working on the papers is the most obvious way to find out what the questions will be. This could be through shoulder surfing while they write the questions, recording them writing the questions on a camera or going through their waste paper bins afterwards in the hope they made notes about the question or drafts which could give an idea of what it will ask about.

Interception

Who's the weakest link? Think of the security economics.

The papers will be passed between a large number of people. Since the questions will not be on networked computers, they must be physically passed between the people making them. Many of these people will be very busy and may not pass the papers directly to the next person, instead leaving them in pigeonholes, at receptions or mailing them. All of these methods are susceptible to interception – monitoring the chief examiners mail, pigeonhole or asking to collect packages at reception might prove effective ways to gain access to exam questions.

Malware

The security policy only prevents computers which store the exam questions on from being connected to the internet. It does not prevent the computers which are storing the exam questions from connecting to the internet before

and after storing the exam questions. So if we infected the computer with malware before the exam question was written, we could store a second copy of the paper and send it away once the computer reconnects to the network. Or we could send it out via covert channels (even though the laptop is not connected to the internet, there are methods to send out information – for example if the laptop is connected to the power socket then we could vary the consumption of electricity to send information or it could play specific sounds at specific volumes to convey bits information [the later method is more likely to be noticed but also has a far higher bandwidth]).

"It does not prevent the computers which are storing the exam questions from connecting to the internet before and after storing them"
Right! So the requirements rae probably poorly phrased and don't reflect the real intention. Actually, the question is expecting you to comment on that.

"Laptop is connected to the power socket then we could vary the consumption of electricity to send information"
Like a budget version of a HomePlug?

"or it could play specific sounds at specific volumes to convey bits of information"
The famous version of this is the TEMPEST attacks, where it was pretty easy to read off a CRT screen even with a wall in the way because of all the electromagnetic interference.

Post-Completion

The questions will be printed several days or weeks prior to the exam. The most obvious way to get them would be to steal a physical copy of one. However, these papers are always closely guarded and so this is unrealistic. A more feasible approach would be to find the printers they're most likely to be printed on and turn on "keep printed documents". The printer would then keep a copy of the documents it has recently printed. Including several hundred copies of exam papers. Accessing it after the printing would give a full copy of the exam questions.

(b) Describe a security policy suitable for department A, taking into account the head-of-department's requirements and the staff workflow. Discuss it thoroughly, including requirements analysis, incentives and technical mechanisms.

- The examination board will distribute encrypted hard drives for the sole purpose of working on exam questions.

  "encrypted hard drives"
  Careful, even in a CS department, your staff may not be very technological. And while this obeys the "reside" request, isn't it an overly literal interpretation of it? Given that the resident malware could just take a copy of it and forward it on later.

- Exam questions must only be stored on these hard drives.
- Hard Drives or their contents must not be given to third parties.
- The hard drives must be returned directly to the exams office for redistribution after work with them has finished.
- Any computer being used to write exam questions must not be connected to any network.

This allows staff to work on questions when they are in many places and allows them to easily redistribute the past questions easily. The use of encrypted hard

drives (USB drives) means that staff can work on the questions, save it back to the USB drive and then reconnect to the internet and continue to use their computers as normal. This is good for staff workflow – they're not writing off their computer for the period they work on the question or are forced to use another sub-par device.

Staff are incentivised to save a copy onto their laptops, work on it and then at the end of writing the question to save it onto the encrypted drive. However, since there are so few precautions, I think the policy is within their compliance budget – staff can still use their favourite text editors and IDEs on their own device and can find information on other devices. Staff have a personal interest for questions not to be released.

"I think the policy is within their compliance budget"
Right! This is important; part (*c*) is basically goading you down this path.

"Staff have a personal interest for questions not to be released"
Yes – most of them at least.

Since questions must be passed directly to the exams office, the risk of interception is low – the exams staff know not to give questions away and they will be stored in secure locations until they're given to the chief examiner or external auditor.

This scheme only requires the university to purchase encrypted hard drives which are resilient to brute force attacks. These are relatively inexpensive.

(c) The Head of Department *B* finds that *A*'s requirement would impose an excessive penalty on the productivity of her staff. At the same time, she certainly does not want to be blamed for the next leak.

(i) Describe a security policy suitable for department *B*, taking into account the head-of-department's requirements and the staff workflow. Discuss it thoroughly, including requirements analysis, incentives and technical mechanisms.

- The chief examiner shall setup password protected drives with basic text editor facilities on the university's internal network for each question. These drives shall only be accessible to the professors and the chief examiners university account on their specific devices.
- Work shall only be done on the exam questions in these drives.
- Questions shall only be accessed in private locations – this is limited to offices, private accommodation and staff-only facilities.
- The chief examiner will choose suitably secure passwords and be responsible for managing who has access rights for each drive.
- Staff working on questions shall have the university's designated antivirus software installed and shall run quick scans daily and full scans weekly.

This sounds a lot like the setup we used at Edinburgh – though Ross couldn't use it so we ended up sending them by email instead... oops!

Staff can still access and work on exam questions whenever they wish, however the questions are not stored on their devices and can only be accessed on the member of staff's device on the university network using their ID and the drive password.

For a malicious attacker (student) to gain access, they'd have to have access to the member of staff's laptop and know their full login credentials and the password to the shared google drive. This would be a significant breach and it would be immediately obvious that it had happened. Additionally since the questions are stored separately until the end, questions can only be

compromised individually – so the worst-case is no longer the whole paper has been compromised – rather a few individual questions.

If a leak occurs, then a users university account will be fully compromised and it will be clear which member of staff was responsible for the leak. Since the leaks are now tied to member of staff's other accounts and responsibility can easily be attributed; there is now an even higher incentive for all members of staff *not* to be responsible for a leak.

The Chief examiner is also responsible: it is their responsibility that only people who have a legitimate right to know what the questions are will see them. Also since the chief examiner sets teh passwords, its their responsibility to choose good passwords (since passwords are distributed, people so we will find out if they don't)!

Since the questions are stored in the network, they can easily be sent between users without any risk of interception: disable the professor from seeing the question and enable the chief examiner. Disable the chief examiner and enable the external auditor.

This protocol requires a special internal system to be designed which can use the information associated with the university network to distinguish devices and users. This should be developed by a competent contractor.

(ii) Describe three trade-offs between security and usability that you considered in devising the policy in (b)(i) and justify the choices you made.

Files are edited on the university network

This decision involves two subtle trade-offs. Firstly, you can now only work on past questions while connected to the university network. This is bad for usability – professors can't work on questions while on the train or on holiday! However, this means that anytime they work on questions, they will be connected to a secure network. This greatly reduces the risk of spyware. This also puts more of the security aspect into the university's control – individual professors can't email questions or leave paper copies lying around! It also means brute force attacks on encrypted or password-protected files are no longer feasible (since the network will add delay and lock out).

"While connected to the university network"
Maybe allow VPN-ing in? Though perimeters are unfashionable anyway these days – if your cleaner can steal the files just because they're in the building then your security may not be that good.

Examiner sets passwords

One of the main problems with passwords is that the passwords people choose are often weak and easily guessable. If the chief examiner sets the passwords then they are guaranteed to be secure – and if they're not then they can be reported by the professors. The use of strong, new passwords is bad for usability – professors will have to write them down. However, this is not a major problem – if the passwords are written down in a professor houses (as is most likely) then getting exam papers would require breaking and entering! Very few students would resort to such measures.

"If the chief examiner sets the passwords then they are guaranteed to be secure"
But if someone else sets the passwords, the professors won't remember them – so they'll write them down as you say.

"would require breaking and entering"
Right! Don't make your defences impenetrable – make them tamper-evident.

Only accessible on Personal Computers

This is a fairly obvious tradeoff. Although it means that staff can't work on questions on library computers, mobile phones or other devices; it also means questions can't be accessed by third parties without stealing the professors personal laptop and knowing it's password – theft of a laptop is highly likely to get noticed!

2. Discuss the contribution and the relative value of the following aspects of the modern development environment. Illustrate with examples if you can. In each case, would you discard this feature if your employer let you, or insist on retaining it (even covertly) should your employer not value it? Explain your reasons.

   (a) Dividing a project into short development episodes or sprints.

   Large projects are split up into smaller sub-projects which can be completed in manageable amounts of time.

   I would insist on continuing this – breaking larger problems into manageable pieces helps control the complexity of large projects and keep programmers motivated (working on an unmanageably large project where you never seem to complete anything is highly demotivating). This also helps you track how much of a large project has been completed.

   Consider building a (small) operating system. We could divide this into "memory management", "device drivers", "UI", "process management" and more. This makes the project more manageable than "build an Operating System".

   Is the 2-week sprint suitable for every project?

   (b) Project progress visualisation tools such as PERT and GANTT charts.

   While it's vitally important to know how far through a project you are, GANTT charts can be very hard to visualise for any nontrivial projects and end up as unintelligible spaghetti! This is unusable and can even be used to confuse management – who don't want to ask how to understand the report their programmers given them.

   PERT charts partially solve this problem, representing projects as directed acyclic graphs (which can be zoomed into and re-adjusted for visibility and clarity). PERT charts also allow critical path analysis which let us see how to redistribute work and view how delays will affect the completion date of the project.

   However, to make a project progress visualisation chart, everything about a project must be known before it's started – including every phase and approximately how long each stage of the project will last. This is usually infeasible. However, we can make approximate charts – which are still helpful.

   Project Progress Visualisation is essential – if we don't know how far along a project is, then we can fool ourselves into thinking we're further than we are! I would therefore insist on using PERT charts in most projects.

   Well, you use PERT for planning to see the critical paths – and base your GANTT schedule on that.

   (c) Automated regression testing tools.

   Automated regression testing tools will run a large set of tests (which have been built up over the systems lifetime) on the full system before any changes are

committed. If any of these tests fail then the new changes have likely reintroduced a bug. These changes will not be committed to the main system.

Automated regression testing makes changing large systems easier and less risky. Finding and debugging old bugs can be especially painful – we know f used to work and so assume it still works, leading to a painful debugging process. I would therefore strongly suggest automated regression testing and would insist on using it even without encouragement from an employer.

Consider designing a database management system. We might add a new feature that interferes and break existing functionality – for example adding a "add empty row" command would break the invariant that primary keys are non-null meaning merges will not work. Imagine now that this is committed and rolls out into the main system. Bug reports come in and a week after the new feature is added, a different programmer is told to fix this bug. They will have to find out why merges are broken, find out when the invariant of non-null primary keys is broken, read a different programmer's code, find out what it's meant to do and where it fails. This is a long and painful process. Automated regression testing could pick this up and prevent this bug.

Agreed – but do you use automated regression testing on your own code?

(d) Source code management tools.

Source code management tools such as Git or GitHub help ensure that everyone's code is consistent and allows people to easily work on code in parallel. This can reduce the complexity of projects since we only talk to one server rather than to every other programmer individually. They also provide functionality to recover previous code and see recent changes. This allows faster recovery from errors. Code repositories also allow people to see all the work other parts of the team are doing and get a better indication of the progress of the project. I would insist on using a code repository for any project I expected to take more than a few hours.

Agreed – but do you use source control for all your own code?

(e) Scrumming.

Scrums are short meetings where everyone debriefs the rest of the team on their current work status: what they've done recently, what they're planning on doing and what problems they've encountered. Only programmers are allowed to talk.

Scrumming encourages teamwork, cooperation, gives the team a good overall idea of the rest of the project and encourages people to work harder – it's embarrassing to consistently tell people you've done hardly anything! I would therefore encourage scrumming in most situations – especially in a remote working environment where teamwork is very difficult and it's easy to be unaware of the progress of other parts of the project.

"it's embarrassing to consistently tell people you've hardly done anything!"
Actually, I've been very busy debugging!

3. 1 2000 Paper 9 Question 6

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2000p9q6.pdf

The owner of a banking system which previously used manually distributed shared keys to compute MACs on transactions decides to use public key cryptography to distribute MAC keys in future. The proposed protocol is

$$A \to B : \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}$$

(a) Explain the symbolism used in this description.

- $A \to B$ means that A sends a message to B
- T_A is a timestamp generated by A when they sent their message
- K_{AB} is the MAC key shared between A and B
- $_K$ means encrypted under the key K
- K_A^{-1} is A's private key
- K_B is B's public key

Overall the protocol is "A sends a message to B containing a timestamp and the MAC key signed by A by encrypting it under A's private key, and then encrypted using B's public key.

yep, "encrypting under A's private key" is signing in this context.

(b) What is wrong with this protocol?

Public key encryption algorithms commute.

So:

$$\{\{M\}_{K_A^{-1}}\}_{K_B} = \{\{M\}_{K_B}\}_{K_A^{-1}}$$

A third party C knows A's public key and K_C^{-1} so if they intercepted the message they could encrypt under both:

$$\begin{aligned}\{\{\{\{M\}_{K_B}\}_{K_A^{-1}}\}_{K_A}\}_{K_C^{-1}} &= \{\{M\}_{K_B}\}_{K_C^{-1}} \\ &= \{\{M\}_{K_C^{-1}}\}_{K_B}\end{aligned}$$

So a third party C could intercept the message, un-sign A's signature and sign it themselves. C could send this message to B and B would now think that it had come from C.

You gave me an unexpected answer, which I had to look up: we can probably assume this is RSA, which doesn't commute. Though public crypto key protocols do exist that do. I haven't checked this in enough detail, but I think that under your assumptions C never gets an unencrypted copy of M (so can't see the secret key) and that even if it did, C setting up a communication with B that just happens to use the same key as $A - B$ doesn't really break anything for A or B?

As a hint of what actually goes wrong here, notice how similar this protocol is to Needham-Schroeder, which has a famous bug…

4. "Software has become more reliable over the past 20 years." To what extent do you agree with this? Write a short bullet point essay plan with factors for and against.

- Over the last 20 years computational capabilities have greatly increased and the field has matured and attracted interest. This has led to significantly larger projects being undertaken. Since modern projects have a higher intrinsic complexity than projects 20 years ago, they've experienced a decrease in reliability.

  "they've experienced a decrease in reliability"
  and yet our tools have gotten better, haven't they?

- Computers are more powerful and compilers are better than 20 years ago. As a result, speed has become a less critical part of programming and programmers no longer need to think about the efficiency of their code to the same extent. Since efficiency is no longer the primary focus when programming, we can focus more on correctness. This has led to an increase in reliability.

- Over the last 20 years, we've gained a lot of experience in large projects. This has led to an increased emphasis on testing: automated regression testing, unit testing and system testing. So a lower proportion of bugs escape development. Experience in large projects has also led to better development methodologies consider agile vs waterfall (although Brooke's "Extreme Programming" was initially released in 1999 the methodology was not common and was in it's infancy). Many of the techniques discussed in 2 are a result of the increased experience (since they've already been discussed, I won't repeat them). This experience has caused software reliability to increase.

  "although Brooke's "Extreme Programming" was initially released in 1999, the methodology was not common and was in it's infancy"
  When I first set this question in 2017, it was less than 20 years ago.

- Corporate attitudes to bug reports have changed; previously when bugs were reported, companies would threaten to call in lawyers to keep the discovery quiet. This attitude has completely changed now and companies actively encourage people to find bugs in their software (and will pay them if they do). This causes many bugs to be quickly found and patched leading to an overall increase in reliability.

  "This attitude has completely changed now and companies actively encourage people to find bugs in their software"
  It's not changed as much as you'd hope – car companies still threaten to sue. Even the tech majors are flaky at paying out bounties – your payday will be more reliable if you go to Pegasus, nation states or worse.

- The languages which are used for development in 2022 have far more features than those which were used 20 years ago. 20 years ago the main development language was C – which is weakly typed and requires manual memory management and meticulous planning. Now the main development languages such as Python, Java or Kotlin are all strongly typed and provide many more features which make development easier.

  "C – which is weakly typed"
  I reckon I'd consider C to be a strongly typed language in comparison to Python. But it's not a term with much meaning. And I'd say that C wasn't the main development language 20 years ago – possibly C++? I think "most popular language" depends on your definition of programming language.

- Over the last 20 years, programming has moved away from "writing it from scratch" to "using the library that does it". For complicated programs, this increases the reliability since mistakes are not made with the intrinsically complicated parts of the code. All we need to do now is call the API .

  However, since programmers have not written much of the complicated code there is often a poor understanding of how it works making it prone to misuse and leading to reliability issues. Giving an example, in Python integers can't overflow. However the main linear algebra library NumPy is written in C – and so can experience integer overflows.

  Overall, the growth of libraries has increased software reliability.

- Computer Science is a more mature discipline. This means there is more interest, literature and education than there was 20 years ago. The effect of this is that there are more good programmers. However, conversely the increased interest and accessibility means it's easy for amateurs to become involved. Overall I'd argue that the increase in quality of programmers due to increased literature and education outweighs the decrease due to accessibility. I'd argue the maturity of Computer Science has resulted in an overall increase in the reliability of software.